INF1010 2016 — Innleveringsoppgave 11 — SUDOKU

Versjon 27. april. Denne versjonen gjelder oppgave 11. Resten er tatt med bare for å se sammenhengen.

Oppgaven er delt inn i tre innleveringer (8, 10 og 11), slik at det er mulig å få tilbakemeldinger underveis. Innleveringsfrister:

- *Oppgave 8 - 13. april*
- *Oppgave 10 - 4. mai*
- *Oppgave 11 - 11. mai*

Oppgave 8 og 10 er igjen delt i to. Arbeidet kan fordeles på en 6-ukers periode med ca. 3 poengs arbeid pr uke. Tilsammen skal du ende opp med et program som kort fortalt løser sudokuoppgaver. Men før vi beskriver de enkelte delene, tar vi en gjennomgang av hele oppgaven for oversiktens skyld.

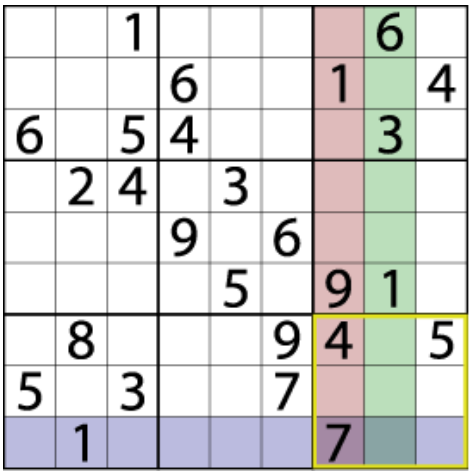
Hva er sudoku?

Se denne wikipediartikkelen:

`https://no.wikipedia.org/wiki/Sudoku`. (For mange fler detaljer, gå til den engelske wikipediaartikkelen `https://en.wikipedia.org/wiki/Sudoku`.)

Sudokubrettet

Et sudokubrett består av n × n ruter. Vi bruker følgende begreper i oppgaven:



- **rute** er feltet som det kan stå ett tall (eller én bokstav) i.
- **brett** er alle $n \times n$ ruter.
- **rad** er en vannrett (fra venstre mot høyre på brettet) rekke med n ruter.
- **kolonne** er en loddrett (ovenfra og nedover) rekke med n ruter.
- **boks** er flere vannrette og loddrette ruter, markert med tykkere strek i oppgavene, ofte med vekslende bakgrunnsfarge; i 9×9-sudoku er en boks på 3×3 ruter, mens i lynsudoku består en boks av 2×3 ruter.

Brettet ovenfor er et 9×9-brett ($n = 9$), dvs. rader, kolonner og bokser har alle 9 ruter. Det er også 9 rader, 9 kolonner og 9 bokser i brettet. 2 kolonner er markert med rød og grønn bakgrunnsfarge. Nederste (niende) rad er markert med fiolett, mens boksen nederst til høyre er rammet inn med gult. Det er vanlig å si at øverste rad er første rad, mens kolonnen lengst til venstre er første kolonne.

Disse begrepene vil i javaprogrammet finnes igjen som objekter. F.eks. vil hele brettet representeres av et objekt som bl.a. inneholder en array av ruter. Hver rute vil igjen inneholde informasjon om hvilken rad, kolonne og boks ruta ligger i.

Du skal lage programmet så generelt at det også kan løse sudokubrett som

ikke har kvadratiske bokser. Brettet ovenfor er eksempel på et brett med kvadratiske bokser. Merk at 64×64 er den øvre grense for størrelsen på brettet. Eksempel på et ikke-kvadratisk brett ser du nedenfor (6×6).

En **sudokuoppgave** er et delvis utfylt brett som kan ha tre løsningsmuligheter:

1. én løsning (slike finner vi i aviser, bøker og blader)
2. ingen løsning (tallene er plassert slik at det ikke finnes en løsning)
3. flere løsninger (for få forhåndsutfylte tall)

Et tomt 9×9 brett har 6.670.903.752.021.072.936.960 løsninger.

Hovedetappene under utførelsen av programmet

- lese inn brettet
- opprette en datastruktur som tilsvarer det innleste brettet
- finne alle løsninger og lagre disse i en beholder
- skrive ut løsningene til skjerm og fil
- skrive ut løsningene én etter én, når brukeren trykker på en knapp

Men utviklingen trenger ikke skje i denne rekkefølgen. Mer avansert GUI, f.eks. kan utvikles uavhengig av resten. Det samme gjelder innlesing fra fil, og beholderen som lagrer løsninger. Datastruktur og metoden som finner løsninger, derimot, henger tett sammen. Vi anbefaler at man begynner med datastrukturen. Dette er av erfaring den delen man trenger lengst tid på. Derfor er løsningen av denne skilt ut som det første du skal gjøre.

Filformatet vi bruker

For å lagre sudokuoppgaver har vi derfinert et eget filformat. Dette 6×6-brettet beskrives av filen til høyre. Merk at filen ikke har blanke tegn i seg:

		3	6		
	2				4
5				6	
	3				5
3				1	
		1	4		

```
2
3
..36..
.2...4
5...6.
.3...5
3...1.
..14..
```

Første tall (2) er antall rader i hver boks, og neste tall (3) er antall kolonner i hver boks. $2 \times 3 = 6$. Brettet er da $6 \times 6 = 36$ ruter. 6 er også antall ruter i kolonne, boks og rad. Så følger selve brettet. Punktum (.) betyr tom rute. Hvis vi trenger mer enn 9 verdier, bruker vi følgende:

Tegn	Verdi
1 - 9	1 - 9
A - Z	10 - 35
@	36
#	37
&	38
a - z	39 - 64

Disse to metodene kan du fritt bruke for å konvertere fra tall til verdi og omvendt.

Programmodulen som leser fra fil og oppretter brettet kan utsettes, men for å få poeng for den, må den seinest tas med i leveringen av oppgave 8.

Del 8a—Les inn fra fil, datastruktur for brett og ruter

Størrelsen brettet og selve oppgaven (de sifrene som alt er fylt inn) leses fra fil. Det er mulig å gjøre selve innlesningsdelen etter del 8b, men for å ha data å jobbe med er det greit å lage en metode som leser inn sudokuoppgaven fra fil med en gang. Den første oppgaven består derfor av å lage en metode som leser filen, bestemmer størrelsen på brett og boks og oppretter datastrukturen for brettet. Under innlesing skal det kastes unntak for disse feilsituasjonene:

- Fil med angitt filnavn eksisterer ikke
- Ugyldig tegn i filen
- Tall utenfor lovlig intervall[1]
- Antall tegn stemmer ikke[2]
- For stort brett (større enn 64×64)

Klassen Brett må inneholde info om alle rutene i sudokubrettet. Info om hver rute lagres i objekter av klassen Rute.

Klassen Rute må inneholde en verdi (husk at noen av rutene ikke har en verdi ennå). For hver rute du leser inn; opprett et objekt av klassen Rute og lagre verdien. Lagre så hver rute i brettet (f.eks. i en array).

Lag så en utskriftsmetode som løper gjennom rutene i brettet ditt og skriver ut verdiene til skjerm. Utskriften fra eksemplet ovenfor skal være slik (uløst til venstre):

```
  3|6                    453|621
 2 |  4                  126|534
---+---                  ---+---
5  | 6                   514|362
 3 |  5                  632|145
---+---                  ---+---
3  | 1                   345|216
  1|4                    261|453
```

Oppsummering: Lag en metode lesFil() som leser inn en fil og lagrer infoen i et objekt av klassen Brett. Lag så en utskriftsmetode som skriver ut det

[1] Lovlige verdier er fra og med 1 til og med produktet av de to første tallene

[2] skal være kvadratet av produktet av de to første tallene i fila (+ 2)

innleste brettet til skjerm.

Del 8b—utvid datastrukturen

For å gjøre det mulig å løse sudokubrettet må klassen Rute i tillegg til verdi inneholde info om hvilken rad, kolonne og boks den tilhører.

Dette gjøres ved at *hver rute* inneholder pekere til ett objekt av klassen Boks, ett objekt av klassen Rad og ett objekt av klassen Kolonne. Ruter i samme rad skal peke på samme radobjekt osv.

Disse klassene skal brukes for å bestemme hvilke verdier som er «opptatt» i raden, kolonnen og boksen. Klassen rad, kolonne og boks må derfor inneholde info om hvilke verdier som allerede er tatt i den raden/kolonnen/boksen.

Lag en metode

```
void opprettDatastruktur()
```

som oppretter riktig antall rader, kolonner og bokser, for så å få hvert ruteobjekt i brettet til å peke på sine respektive rader, kolonner og bokser. For at inndelingen i bokser skal bli riktig må du ha tatt vare på de 2 første verdiene i fila som sier antall rader i hver boks og antall kolonner i hver boks.

Tegn datastrukturen til ruta i 3. rad og 3. kolonne i figuren under beskrivelsen av filformatet. *Denne tegningen skal leveres i Devilry med mindre din retter har gitt fritak i forbindelse med retting.*

Skriv tilslutt en metode i klassen Rute

```
int[] finnAlleMuligeTall()
```

som returnerer en array med de tallene som er mulige løsningstall i en blank rute. F.eks. skal metoden kalt i øverste rute til venstre i eksempelbrettet på første side returnere en array med tallene 2, 3, 4, 7, 8 og 9.

Oppsummering: Utvid klassen Rute til å inneholde info om rad, kolonne og boks. Lag så metoden int[] finnAlleMuligeTall(). Husk tegning av datastrukturen til en rute.

Tips: Gi hver rad/kolonnoe/boks en unik ID, på denne måten kan du enkelt lage en testutskrift som løper gjennom brettet og skriver ut hvilken rad/kolonne/boks hver rute tilhører. Da ser man fort om man har gjort en feil. *Eller:* Lag en testmetode som løper igjennom alle rutene og som for hver rute skriver ut info om verdi, rad, kolonne og hvilke tall som er mulige løsningstall (ved kall på finnAlleMuligeTall)

Del 10a—løsningsmetoden og utskrift av løsninger

Løsningene skal finnes ved å gå gjennom alle rutene på brettet og prøve alle mulige (lovlige) verdier i hver eneste rute. Dette kalles «rå kraft»-metode («brute-force» på engelsk).

Utvid klassen Rute med en metode, fyllUtDenneRuteOgResten, som prøver å sette alle mulige løsningstall i seg selv. Det aller første denne metoden gjør er å kalle på metoden finnAlleMuligeTall. Deretter prøver den å sette alle disse tallene i denne ruta, ett om gangen. For hvert tall som settes i ruta kalles samme metode (fyllUtDenneRuteOgResten) i neste rute (dvs den rett til høyre). Når en vannrett rad er ferdig (det finnes ingen rute rett til

høyre), kalles metoden i ruta helt til venstre i neste rad, osv. Når et kall på fyllUtDenneRuteOgResten-metoden i neste rute returnerer, prøver ruta neste tall som enda ikke er prøvd, osv. helt til alle tall er prøvd i denne ruta. Main-metoden starter det hele ved å kalle fyllUtDenneRuteOgResten i den øverste venstre ruta.

Løsningen(e) skal skrives ut på skjermen.

Noen hint:

• Du kan gjerne lenke sammen alle rutene med en nestepeker, slik at en rute bare kan kalle neste.fyllUtDenneRuteOgResten.

• Skriv en metode i Brett som finner neste tomme rute og kall metoden i denne). Når metoden har funnet en lovlig verdi i den siste tomme ruta (vanligvis den lengst nederst til høyre) på brettet, er en løsning funnet.

• Legg alle de tomme rutene inn i en beholder, ta dem ut en etter en og kall løsningsmetoden i disse.

• Under utviklingen kan det være lurt å først lage et program som genererer alle løsninger for et tomt brett for så senere å utvide med at noen av rutene kan ha forhåndsutfylte verdier. Ikke bruk et tomt brett med mer enn 9×9 ruter, da dette tar fryktelig lang tid.

For dem som ønsker å forbedre programmet utover minimumskravene

Metoden over (brute-force) er enkel og gjør jobben, men kan ta fryktelig lang tid for store brett. Det understrekes at forbedringene nedenfor *ikke* er nødvendig for å kunne få full pott på del 10. Det anbefales at man ikke går igang med dette før man har en tilfredstillende løsning med brute-force og en løsning av del 11. Her er noen tips for å forbedre hastigheten:

• Fyll inn ruter med få muligheter først, særlig bør ruter med én mulighet fylles inn umiddelbart da dette vil eliminere mange andre muligheter. Det går an å ordne rutene først etter stigende antall muligheter (kandidattall), f.eks. ved å legge tomme ruter inn i en beholder og så ta dem i den rekkefølgen uavhengig av hvordan antall muligheter i rutene skulle endre seg underveis. Det siste kan man også ta hensyn til, men krever mye mer «bokholderi» underveis.

• Det kan være raskere å sjekke alle rutene aller først og lagre de mulige verdiene i et boolean-array, og så kan man bruke noen «penn-og-papir-metoder» for å eliminere en del av dem før man eventuelt faller tilbake på brute-force.

• Det går an å bruke tråder til å finne løsninger i parallell.

Før du går i gang med å forbedre programmet utover minimumskravene, bør du ta en kopi (for levering) i tilfelle du får problemer med å fullføre det innen fristen.

Del 10b—utskrift av løsninger

Filnavnet oppgis som parameter til programmet (på kommandolinja). Filformatet skal være slik som beskrevet over (retter vil teste programmet ditt med andre filer). Løsningen(e) skrives til skjerm.

Filformatet for utskrift er beskrevet nedenfor.

Programmet skal inneholde class SudokuBeholder som igjen inneholder de tre offentlige metodene settInn, taUt og hentAntallLosninger. Du kan lage beholderklassen selv (da lærer du mer), men kan også gjenbruke klassen fra en tidligere oppgave, eller bruke en klasse fra Javas API.

Programmet skal finne løsninger og legge dem inn i et objekt av klassen SudokuBeholder. Husk å ta en kopi av løsningen og legg kopien inn i beholderen, så unngår du at løsningsmetoden «ødelegger» løsningen. Hvis det finnes flere løsninger enn 3500, skal beholderen holde orden på hvor mange løsninger som

er funnet, men ikke ta vare på flere enn 3500.

Oppgave 11—framvisning av løsninger med vindusbasert GUI

I denne skal du lage et grafisk brukergrensesnitt (GUI) med JavaFX for å kommunisere bedre med brukeren og for å skrive ut løsninger. 3 nye krav:

1. For den grafiske representasjonen av brettet skal programmet bruke klassen javafx.scene.layout.GridPane.

2. Programmet skal bruke klassen javafx.stage.FileChooser for å la brukeren finne/velge filen med oppgaven.

3. Når brukeren har valgt sudokufilen som skal løses, skal løsningen(e) vises ved å hente den/dem fra beholderen etter at oppgaven er løst. Løsningen(e) kan vises i samme vindu, eller det kan åpnes et nytt vindu med løsningen(e). Er det flere enn en løsning vises bare den første fram, med informasjon om hvor mange løsninger som totalt ble funnet.

I den avsluttende delen skal du utvide GUI-et med lyttende knapper slik at løsningene (hvis flere) kan vises fram en etter en når brukeren trykker på en knapp. Her kan du også legge inn ande nyttige GUI-funksjoner som f.eks. muligheten til å legge inn en oppgave direkte i vinduet.

Om tråder

I denne oppgaven trenger du ikke programmere med andre tråder enn maintråden og GUI-tråden. Maintråden starter med å opprette et eller flere vinduer for å lese inn data. Disse vinduene skal så lukkes når data er lest inn. Da startes selve løsningsalgoritmen. Når maintråden har funnet en løsning legges denne inn i en beholder. Når alle løsningene er funnet skal maintråden åpne et vindu der brukeren kan be om at en og en løsning blir vist fram ved å trykke på en knapp i vinduet.

Prøv å lage et robust program, dvs. et som ikke kræsjer når filformatet er feil eller noe annet uventet skjer.

Om du synes at noen av disse kravene er urimelige, eller du synes du kan løse oppgaven mer elegant eller bedre på en annen måte, ta det opp med din retter i forbindelse med rettingen av de tidlige delene. En forutsetnig for å gjøre dette, er at man har møtt til retting underveis.

Om formatet for utskrift til skjerm

Formatet er ganske likt innfilformatet. Punktum byttes ut med løsningstallet. Når det er mange løsninger kan du bruke det alternative utformatet som vises nedenfor. Dette er ikke et krav. Denne sudokuoppgaven har 28 løsninger, de første 9 av dem er listet opp til høyre i alternativt utformat:

```
2
3
..1..3
......
....2.
26....
...3..
3..1.2
```

```
1: 421563//653214//134625//265431//512346//346152//
2: 421653//536214//143526//265431//612345//354162//
3: 421653//536214//153426//264531//612345//345162//
4: 421653//635214//513426//264531//142365//356142//
5: 451263//623415//135624//264531//512346//346152//
6: 451263//623514//134625//265431//512346//346152//
7: 451263//632415//513624//264531//125346//346152//
8: 521463//643215//135624//264531//412356//356142//
9: 521643//436215//143526//265431//612354//354162//
```

Retting underveis

Du bør ukentlig diskutere med gruppelærer/retter om hvordan du skal løse denne oppgaven. Pass på å hele tiden ha et program som kompilerer og kjører (men som i starten ikke gjør særlig mye). Du bør i det minste kontakte dem for å få tilbakemelding etter hver av de fem delene.

I tillegg kan det være greit å få tilbakemeldinger:

• når du har laget main og mange tomme klasser og har en første grove skisse av hele programmet ditt

• når du har bestemt formatet på løsningene slik de skal lagres i sudokubeholderen

• når du har laget en skisse av klassen Rute og dens subklasser

• når du har laget en skisse av klassene Boks, Kolonne og Rad og superklassen til disse klassene.

• når du har laget en skisse av GUI-programmet som henter ut løsninger fra sudokubeholderen og tegner dem ut.

Flere eksempler, brett i forskjellige størrelser, eksempler på brett laget med Swing/JavaFX, flere sudokuoppgaver, samt utfyllende informasjon om sudoku finner du på INF1010s Sudoku-side:

`http://heim.ifi.uio.no/inf1010/v16/innleveringsoppgave/08_10_11/sudoku.html`

Om alternativ datastruktur

Hvis du ønsker å ha variable i klassen Rute som kan brukes til å midlertidig begrense gyldige verdier i denne ruta, eller gjøre andre større endringer i datastruktur, skal du først diskutere dette med din retter og få tillatelse.

Programmeringskonkurranse

Det vil være tre kategorier i konkurransen:

1. Raskeste løsning (her må deltageren redegjøre for hvilke løsningsmetoder som er brukt og hvordan de virker)
2. Beste brukeropplevelse (her vil et pent, oversiktlig og brukervennlig GUI være viktig)
3. Åpen klasse. For deg som har gjort noe ekstra som du synes er kult (en idé kan være å lage totalt unyttig ekstrafunksjonalitet).

Bidrag leveres i Devilry i mappen som heter «Sudokukonkurranse». Frist for konkurransebidrag er 23. mai kl. 10.00.

Innleveringen skal i tillegg til de relevante .java-filene inneholde en README-fil der følgende kommer frem:

• Hvilke kategorier du ønsker å konkurrere i

• En beskrivelse av hva du har gjort

• En beskrivelse av hvordan programmet skal kjøres/brukes

Det blir premiering til deltakerne og kåring av vinnere ca. en uke før eksamen.