

MAT1030 – Diskret matematikk
Forelesningsnotater
Våren 2008

Dag Normann Roger Antonsen
Universitetet i Oslo



Sist oppdatert: 27. mai 2008

Dette kompendiet er automatisk generert fra materialet som ble presentert på forelesningene og plenumsregningene, og må leses med dette forbeholdet.

Kommentarer, feil og forslag til forbedringer kan sendes til:

Roger Antonsen (rantonse@ifi.uio.no) eller Dag Normann (dnormann@math.uio.no).

Innhold

Forelesning 1: Algoritmer, pseudokoder og kontrollstrukturer (14.01.2008)	5
Forelesning 2: Flere pseudokoder. Representasjoner av tall. (16.01.2008)	13
Forelesning 3: Mer om representasjon av tall (21.01.2008)	21
Forelesning 4: Tall som data (23.01.2008)	29
Forelesning 5: Logikk (28.01.2008)	41
Forelesning 6: Logikk (30.01.2008)	53
Forelesning 7: Predikatlogikk (04.02.2008)	63
Forelesning 8: Predikatlogikk, bevisføring (06.02.2008)	75
Forelesning 9: Mengdelære (11.02.2008)	87
Forelesning 10: Mengdelære (13.02.2008)	99
Forelesning 11: Relasjoner (18.02.2008)	113
Forelesning 12: Relasjoner, Funksjoner (20.02.2008)	123
Forelesning 13: Funksjoner (25.02.2008)	137
Forelesning 14: Rekursjon og induksjon (27.02.2008)	149
Forelesning 15: Induksjon og rekursjon, rekurenslikninger (03.03.2008)	159
Forelesning 16: Rekurrenslikninger (05.03.2008)	171
Forelesning 17: Generell rekursjon og induksjon (10.03.2008)	187
Forelesning 18: Generell rekursjon og induksjon (12.03.2008)	199
Forelesning 19: Kombinatorikk (26.03.2008)	211
Forelesning 20: Kombinatorikk (07.04.2008)	225
Forelesning 21: Grafteori (09.04.2008)	237
Forelesning 22: Grafteori (14.04.2008)	249
Forelesning 23: Grafteori, Trær (16.04.2008)	263
Forelesning 24: Trær (21.04.2008)	273
Forelesning 25: Trær (23.04.2008)	281
Forelesning 26: Trær (28.04.2008)	289
Forelesning 27: Trær (30.04.2008)	297
Forelesning 28: Grafer og trær, eksempler (05.05.2008)	307
Forelesning 29: Kompleksitetsteori (07.05.2008)	313
Forelesning 30: Kompleksitetsteori (14.05.2008)	319
Forelesning 31: Kompleksitetsteori (19.05.2008)	329
Forelesning 32: Repetisjon (21.05.2008)	341
Forelesning 33: Repetisjon (26.05.2008)	349

Plenumsregning 1: Kapittel 1 (17.01.2008)	357
Plenumsregning 2: Ukeoppgaver fra kapittel 1 & 2 (24.01.2008)	365
Plenumsregning 3: Ukeoppgaver fra kapittel 2 & 3 (31.01.2008)	377
Plenumsregning 4: Ukeoppgaver fra kapittel 3 & 4 (07.02.2008)	385
Plenumsregning 5: Ukeoppgaver fra kapittel 4 (14.02.2008)	393
Plenumsregning 6: Ukeoppgaver fra kapittel 5 (21.02.2008)	403
Plenumsregning 7: Ukeoppgaver fra kapittel 5 & 6, mm. (28.02.2008)	415
Plenumsregning 8: Diverse ukeoppgaver fra kapittel 6 & 7, mm. (27.03.2008)	427
Plenumsregning 9: Diverse ukeoppgaver (10.04.2008)	435
Plenumsregning 10: Diverse ukeoppgaver (17.04.2008)	445
Plenumsregning 11: Ukeoppgaver fra kapittel 10 & Induksjonsbevis (24.04.2008)	453
Plenumsregning 12: Diverse oppgaver (22.05.2008)	459

Forelesning 1

Algoritmer, pseudokoder og kontrollstrukturer

Dag Normann - 14. januar 2008

Vi som skal undervise

- Dag Normann
 - Roger Antonsen
 - Christian Schaal
 - Robin Bjørnetun Jacobsen
- <http://www.uio.no/studier/emner/matnat/math/MAT1030/v08/>

Hva er diskret matematikk?

- Diskret matematikk er et samlebegrep for matematikk hvor kontinuitet, geometri eller algebra ikke spiller noen stor rolle.
- Diskret matematikk er matematikken tilpasset en digital verden, mens mye annen matematikk er tilpasset en analog verden.
- I MAT1030 skal vi ta for oss temaer som er relevante for en grunnleggende forståelse av bruk av, og virkemåte til, datamaskiner.

Hva er innholdet i MAT1030?

- Algoritmer: Hva og hvordan.
- Representasjon av tall.
- Hvordan representeres tall og i en datamaskin.
- Logikk:
 - Relevans for informatikk.
 - Innføring i utsagnslogikk (Boolesk logikk).
 - Litt om bevisteknikker.
- Mengdelære:
 - Grunnleggende begreper
 - Relasjoner.
 - Funksjoner.
- Induksjon og rekursjon.
- Kombinatorikk.
- Grafteori med anvendelser.

- Litt om trær.
- Kompleksitet av algoritmer.

Hva legger vi vekt på?

- Hvilken relevans har dette stoffet for studier i informatikk?
- Hvordan kan vi finne algoritmer for å løse de problemene som presenterer seg?

Vi kommer til å arbeide med algoritmer i tilknytning til logikk i større grad enn det boka gjør, og vi vil utvikle en del ekstra undervisningsmateriale underveis.

Før vi begynner

- MAT1030 er under omlegging, og vi trenger en kontinuerlig tilbakemelding på årets opplegg.
- Vi trenger to eller flere kontaktpersoner vi kan ha møter med gjennom semesteret.
- Vi skal gjennomføre en emneevaluering i samarbeid med kontaktpersonene.
- Valg av kontaktpersoner finner sted neste onsdag.

Noen viktige datoer

- 29.02.2008: Innlevering av Oblig 1.
- 25.04.2008: Innlevering av Oblig 2.
- 10.06.2008, 09.00 - 12.00: Eksamen.

Viktig informasjon til alle studenter

Husk:

1. februar er siste frist for:

- betaling av semesteravgift
- semesterregistrering
- melding til eksamen
- å søke om tilrettelegging ved eksamen

Dersom du har gjort alt riktig skal det se slik ut i studentweb:

Betalt semesteravgift:	Vår 2008
Semesterregistrert:	Vår 2008

Dersom det ikke står "Vår 2008" under henholdsvis "Betalt semesteravgift" og "Semesterregistrert" risikerer du å ikke få ta eksamen og å miste din studierett.

Har du problemer, ta kontakt med MN studieinfosenter i Vilhelm Bjerknes' hus FØR 1. februar.
Telefon: 22 85 63 44
e-post: studieinfo@matnat.uio.no

ALGORITMER

En algoritme er en oppskrift som forteller oss hvordan vi skritt for skritt skal kunne oppnå et resultat eller løse et problem.

Eksempler på algoritmer kan være:

- Kakeoppskrifter.
- Automatisk innsjekking på fly.
- Beskrivelsen av hvordan man utfører divisjon mellom flersifrede tall.
- Oppskrift på hvordan man løser opp parenteser og trekker sammen flerleddede uttrykk i algebra.

Algoritmer

Hva er det som kjennetegner en algoritme?

Det skal ikke kreves intelligens eller forståelse for å følge den.

- Du skal ikke kunne kjemi for å bake en kake.
- Du skal kunne sjekke inn på fly selv om du har teknologifobi.
- Det er ikke nødvendig å forstå hva man gjør når man utfører en divisjon, regnetrening er det som trengs.
- Mange lærer seg hvordan de kan løse oppgaver i skolealgebra, uten å ha peiling på hva de egentlig driver med.

Algoritmer

Vi skal fokusere på algoritmer som

- beregner funksjoner
- avgjør om et objekt eller en datamengde har en gitt egenskap eller ikke
- organiserer gitte data på en ønsket måte (eksempelvis ordner dataene)
- utfører andre oppgaver i tilknytning til matematikk eller informatikk som vi ønsker å kunne få utført.

Algoritmer

Hvem ønsker vi å kommunisere algoritmen til, og hvordan skal det gjøres?

1. Kakebakere, flypassasjerer og liknende.
2. Skolebarn/ungdom og studenter som skal lære matematikk.
3. Teknisk kyndige medmennesker som skal "forstå" algoritmen.
4. Datamaskiner som skal utføre algoritmen for oss.

I MAT1030 er gruppe 3 den mest aktuelle.

PSEUDOKODER

En pseudokode er en måte å beskrive en algoritme på.

Pseudokoden beskriver hvordan algoritmen kan følges trinn for trinn.

En pseudokode formuleres i et språk som er mer teknisk enn naturlige språk og mindre teknisk enn programmeringsspråk.

Vi skal bruke pseudokoder på samme måte som i læreboka.

Man må se eksempler, og øve, for å bli flink til å skrive pseudokoder.

Pseudokoder

Eksempel (Areal av trekant).

1. Input h [h er høyden i trekanten.]
2. Input g [g er lengden på grunnlinjen i trekanten.]
3. $\text{areal} \leftarrow \frac{h \cdot g}{2}$
4. Output areal

Pseudokoder

Vi kunne ha skrevet en annen pseudokode for å beregne det samme:

Eksempel.

1. Input h
2. Input g
3. $\text{areal} \leftarrow h \cdot g$
4. $\text{areal} \leftarrow \frac{\text{areal}}{2}$
5. Output areal

Pseudokoder

Så langt består en pseudokode av en nummerert liste instruksjoner hvor hver instruksjon har et av følgende tre formater:

- Gi en input-verdi til en variabel.
- Gi en variabel en ny verdi, som en funksjon av de eksisterende verdiene på variablene.
- Gi verdien til en av variablene som output, det vil si resultatet av algoritmen.

Vi kan bruke hva vi vil som variable, eksempelvis er h , g og areal variablene i pseudokodene vi har sett på.

Pseudokoder

Hvis vi skal beregne verdien av en formel for areal, volum, hastighet etter en viss tids fritt fall og liknende, kan vi bruke pseudokoder slik vi har sett dem til nå.

Det finnes imidlertid algoritmer, og tilhørende kontrollstrukturer, for å beregne

- $|x|$ fra x
- $n!$ fra n
- Ledd nummer n i Fibonacci-følgen

1, 1, 2, 3, 5, 8, 13, 21, . . .

og for å undersøke om

- Parentesene i et algebraisk uttrykk er satt på lovlig måte
- Et naturlig tall er et primtall

KONTROLLSTRUKTURER

En kontrollstruktur brukes for å styre hvordan, og hvorvidt, de enkle instruksjonene i en pseudokode skal utføres.

Vi skal innføre de kontrollstrukturene det er aktuelt å bruke i dette emnet via eksempler på bruk. Disse eksemplene skal supplere eksemplene fra læreboka.

Vi skal benytte oss av de samme kontrollstrukturene som boka.

Kontrollstrukturer

Eksempel (Absoluttverdi).

Vi skal gi en pseudokode for å beregne absoluttverdien til et tall x :

- 1 Input x
- 2 **If** $x < 0$ **then**
 - 2.1 $x \leftarrow -x$
- 3 Output x

Kontrollstrukturer

Eksempel (Avstand).

Vi skal gi en pseudokode for å beregne avstanden mellom to heltall.

```
1 Input n [n et heltall]
2 Input m [m et heltall]
3 If n < m then
    3.1 x ← m - n
    else
    3.2 x ← n - m
4 Output x
```

Kontrollstrukturer

Vi skal se tre eksempler på hvordan vi kan skrive pseudokoder for algoritmer som beregner

$$n! = 1 \cdot 2 \cdot \dots \cdot n.$$

I det ene eksemplet bruker vi en while-løkke, i det andre en repeat-until-løkke og i det siste en for-løkke.

Kontrollstrukturer

Eksempel (while-løkke).

```
1 Input n [n ≥ 1, n heltall]
2 x ← 1
3 i ← 1
4 While i ≤ n do
    4.1 x ← x · i
    4.2 i ← i + 1
5 Output x
```

Kontrollstrukturer

Eksempel (repeat-until-løkke).

```
1 Input n [n ≥ 1, n heltall]
```

```
2  $x \leftarrow 1$ 
3  $i \leftarrow 1$ 
4 Repeat
  4.1  $x \leftarrow x \cdot i$ 
  4.2  $i \leftarrow i + 1$ 
  until  $i = n + 1$ 
5 Output  $x$ 
```

Kontrollstrukturer

Eksempel (for-løkke).

```
1 Input  $n$  [ $n \geq 1$ ,  $n$  heltall]
2  $x \leftarrow 1$ 
3  $i \leftarrow 1$ 
4 For  $i = 1$  to  $n$  do
  4.1  $x \leftarrow x \cdot i$ 
5 Output  $x$ 
```

Kontrollstrukturer

Bytte av verdi på variablene, hvordan vi ikke skal gjøre det og hvordan vi skal gjøre det.

Eksempel (Feil måte).

```
1  $y \leftarrow x$ 
2  $x \leftarrow y$ 
```

Eksempel (Riktig måte).

```
1  $hjelp \leftarrow x$ 
2  $x \leftarrow y$ 
3  $y \leftarrow hjelp$ 
```


Forelesning 2

Flere pseudokoder. Representasjoner av tall.

Dag Normann - 16. januar 2008

KONTROLLSTRUKTURER

Mandag innførte vi pseudokoder og kontrollstrukturer. Vi hadde tre typer grunn-instruksjoner:

- Input variabel
- variabel \leftarrow term
- Output variabel

Kontrollstrukturer

Vi hadde fem kontrollstrukturer

- **If** ... **then**
- **If** ... **then** ... **else**
- **While** ... **do**
- **Repeat** ... **until**
- **For** ... **to** ... **do**

Vi skal se på noen flere eksempler på pseudokoder.

Kontrollstrukturer

Det er ingen som vet om algoritmen som er beskrevet i den neste pseudokoden vil terminere for alle input.

Det betyr at den muligens **ikke** er en algoritme i bokas forstand.

Den forutsetter at vi kan skille mellom partall og oddetall.

Eksempel (Ubegrenset while-løkke).

```
1 Input x [x ≥ 1 heltall.]
2 While x > 1 do
  2.1 if x partall then
    2.1.1 x ←  $\frac{x}{2}$ 
  else
```

2.1.2 $x \leftarrow 3x + 1$
3 Output x

Kontrollstrukturer

Vi kan finne en enkel pseudokode for å finne ledd nr. n i følgen

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

av Fibonacci-tall:

Eksempel (Fibonacci).

```
1 Input  $n$  [ $n \geq 1$  heltall]
2  $x \leftarrow 1$ 
3  $y \leftarrow 1$ 
4 For  $i = 2$  to  $n$  do
    4.1  $z \leftarrow x$ 
    4.2  $x \leftarrow x + y$ 
    4.3  $y \leftarrow z$ 
5 Output  $x$ 
```

Kontrollstrukturer

Parenteser i forskjellige former brukes mye i matematikk, informatikk og spesielt i programmer.

Eksempelvis, under utarbeidelsen av denne beamer- presentasjonen forekommer “parenteser” som

`\begin{center}... \end{center}`

og

`\begin{eksempel}... \end{eksempel}`

en rekke steder, og det er viktig at de står riktig i forhold til hverandre.

Dette kontrolleres når råmanuskriptet kompiles.

Det neste eksemplet på en pseudokode er en parentes-sjekker.

Kontrollstrukturer

Vi kan kontrollere om en liste venstre og høyreparenteser

$((()((())))(()((())))(())$

er lovlig eller ikke, ved å telle opp eller ned - opp ved (og ned ved) - fra venstre mot høyre.

Hvis vi til slutt ser at vi har like mange parenteser av hvert slag, og aldri underveis har flere) enn (, er uttrykket i orden.

I den neste eksemplet gir vi to input, lengden av uttrykket og sekvensen av parenteser.

Vi sjekker uttrykket fra venstre mot høyre.

Vi bruker variabelen val til å holde orden på om sekvensen s å langt er i orden.

Vi bruker variabelen y til å telle overskuddet av (.

Kontrollstrukturer

1 Input n [Lengden av uttrykket, antall parenteser totalt]

2 Input $x_1 \cdots x_n$ [En liste av venstre og høyreparenteser]

3 $y \leftarrow 0$

4 val \leftarrow JA

5 For i = 1 to n do

5.1 if $x_i = ($ then

5.1.1 $y \leftarrow y + 1$

else

5.1.2 if $y = 0$ then

5.1.2.1 val \leftarrow NEI

else

5.1.2.2 $y \leftarrow y - 1$

6 if $y > 0$ then

6.1 val \leftarrow NEI

7 Output val

OVER TIL KAPITTEL 2

TALLMENGDER

Hvilke tall vi betrakter er avhengig av hva vi ønsker å bruke dem til.

I MAT1030 vil vi stort sett betrakte følgende typer tall:

- Naturlige tall \mathbb{N}

$1, 2, 3, \dots$

- Hele tall \mathbb{Z}

$\dots, -3, -2, -1, 0, 1, 2, \dots$

- Rasjonale tall \mathbb{Q}

Tall som kan skrives som en brøk $\frac{p}{q}$

- Reelle tall \mathbb{R}

“alle tallene”

Tallmengder

Mange mener at tall er punkter på tall-linja, og at det ikke spiller noen rolle om vi betrakter 2 som et naturlig tall, et heltall, et rasjonalt tall eller et reelt tall.

I programmeringsammenheng kan det spille en stor rolle hva slags verdier en variabel kan få lov til å ta, og representasjonen av et tall som et dataobjekt kan variere med hva slags type tall vi betrakter.

Dette skal vi se nærmere på siden.

Tallmengder

Det finnes andre tallmengder som også er av interesse i matematikk og informatikk, eksempelvis

- Komplekse tall
- Algebraiske tall
- Kvaternioner

REPRESENTASJON AV TALL

Så langt tilbake vi har informasjon om, har mennesker og kulturer hatt muntlig og skriftlig språk for tall.

Romertallet

MCMXXVIII

er en alternativ måte å skrive

1928

på.

Hvis vi blir bedt om å skrive et program for addisjon av to tall, betyr det mye om vi bruker den romerske eller dagens måte å skrive tall på.

Representasjon av tall

De tallene vi bruker til daglig kalles desimaltall, eller tall i 10-tallsystemet.

Dette er et plass-siffersystem med basis 10.

Det betyr igjen at hvert siffer angir et antall 10^i er potenser, og sifferets posisjon forteller oss hvor stor potensen er.

Representasjon av tall

Eksempel.

a) 258 står for

$$2 \cdot 10^2 + 5 \cdot 10^1 + 8 \cdot 10^0.$$

b) 3,14 står for

$$3 \cdot 10^0 + 1 \cdot 10^{-1} + 4 \cdot 10^{-2}$$

Tverrsumtesten

Tverrsummen til et desimaltall er summen av alle sifrene.

Eksempel.

- Tverrsummen til 234 er $2 + 3 + 4 = 9$
- Tverrsummen til 15987 er $1 + 5 + 9 + 8 + 7 = 30$
- Tverrsummen til 2825 er $2 + 8 + 2 + 5 = 17$

Legg merke til at resten vi får når vi deler tallet på 9 er det samme som vi får når vi deler tverrsummen på 9.

Kan dette forklares matematisk?

Tverrsumtesten

Påstand (Tverrsumtesten).

Hvis vi skriver et tall n på desimalform og lar $T(n)$ være tverrsummen til n , så får vi samme rest når vi deler n på 9 som når vi deler $T(n)$ på 9.

Tverrsumtesten

Bevis.

La

$$a_k \dots a_0$$

være desimalformen til n . Da er

$$n = a_k 10^k + \dots + a_1 10 + a_0$$

Hvis $1 \leq i \leq k$ kan $10^i - 1$ deles på 9, siden sifrene består av bare 9-tall.

Vi har at

$$n = T(n) + a_k(10^k - 1) + \dots + a_1(10 - 1)$$

(trenger litt ettertanke), og påstanden følger (trenger litt ettertanke til).

BINÆRE TALL

Det er kulturelt betinget at vi bruke 10 som basis i tallsystemet vårt.

Alle tall > 1 kan i prinsippet brukes.

I informatikksammenheng er det like naturlig å bruke 2, 8 og 16 som basistall.

Bruker vi 2 som basis, sier vi at tallet er på binær form.

Binære tall

Eksempel.

Vi tolker en binær form (hvor alle sifrene er 0 eller 1) omtrent som om det var et desimaltall, bortsett fra at vi erstatter 10 med 2:

- $1010_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 8 + 2 = 10_{10}$
- $11011_2 = 16 + 8 + 2 + 1 = 27$
- $100111001_2 = 256 + 32 + 16 + 8 + 1 = 313$

Binære tall

Binære tall er viktige i informatikk fordi digitalisering innebærer at all informasjon skal lagres som 0'ere og 1'ere, i bits.

Vi skal komme tilbake til hvordan tall representeres i datamaskiner, men bruk av binær representasjon er helt sentral der.

Binære tall

Binær representasjon kan selvfølgelig også brukes til desimaltall.

- $0,100101_2 = \frac{1}{2} + \frac{1}{16} + \frac{1}{64} = \frac{32+4+1}{64} = \frac{100101_2}{64}$
- $0,01101_2 = \frac{1}{4} + \frac{1}{8} + \frac{1}{32} = \frac{8+4+1}{32} = \frac{01101_2}{32}$

Binære tall

Det finnes en enkel prosedyre for å regne ut verdien av et binært tall:

- 1 Input n [Antall sifre]
- 2 Input $x_1 \dots x_n$ [en sekvens av 0'ere og 1'ere]
- 3 $y \leftarrow 0$ [Skal bli verdien på sekvensen tolket som et binært tall]
- 4 **For** $i = 1$ **to** n **do**
 - 4.1 $y \leftarrow 2y$
 - 4.2 **If** $x_i = 1$ **then**
 - 4.2.1 $y \leftarrow y + 1$
- 5 Output y

Vi gir et regneeksempel, med input $n = 4$ og $x_1x_2x_3x_4 = 1101$, på tavla.

Binære tall

Det finnes også prosedyrer for å regne ut verdien av binære tall på desimalform.

Vi skal se på binære tall på formen $0,100110111000101$ og liknende, og hvilken algoritme vi kan bruke for å finne verdien av $0,100110111000101_2$.

- 1 Input n
- 2 $e \leftarrow 1$
- 3 **For** $i = 1$ **to** n **do**
 - 3.1 $e \leftarrow \frac{e}{2}$
- 4 Input $x_1 \dots x_n$ [Det binære tallet som står bak komma]
- 5 $y \leftarrow 0$
- 6 **For** $i = 1$ **to** n **do**
 - 6.1 $y \leftarrow 2y$
 - 6.2 **If** $x_i = 1$ **then**
 - 6.2.1 $y \leftarrow y + 1$
- 7 $y \leftarrow y \cdot e$
- 8 Output y

Binære tall

Det er selvfølgelig viktig å kunne finne verdien av et tall skrevet på binær form.

Det er også viktig å kunne finne binærformen til et tall.

Vi skal se på to pseudokoder, en som finner binærformen til et heltall, og en som finner binærformen til et tall mellom 0 og 1.

I begge tilfeller vil vi finne siffer nr. k fra komma.

Binære tall

Binærformen til et heltall: Skriver ut siffer nr. k i binærformen til n .

```
1 Input n
2 Input k
3 For i = 1 to k do
    3.1 If n like tall then
        3.1.1  $n \leftarrow \frac{n}{2}$ 
        3.1.2  $z \leftarrow 0$ 
    else
        3.1.3  $n \leftarrow \frac{n-1}{2}$ 
        3.1.4  $z \leftarrow 1$ 
4 Output z
```

Forelesning 3

Mer om representasjon av tall

Dag Normann - 21. januar 2008

Oppsummering av Uke 3

Mandag 14.01 og delvis onsdag 16.01 diskuterte vi hva som menes med en algoritme, og vi så på pseudokoder og de tilhørende kontrollstrukturene.

Forventede ferdigheter er:

- Kunne uttrykke en algoritme i pseudokode, og kunne bruke de forskjellige kontrollstrukturene på riktig måte.
- Kunne følge en algoritme gitt ved en pseudokode og input-verdier på variablene skritt for skritt, og kunne holde orden på hvordan verdiene på variablene endrer seg under "utregningen".
- Kunne forklare hvorfor en pseudokode løser den oppgaven den er satt til å utføre.

Oppsummering av Uke 3

Onsdag snakket vi generelt om tall og tallsystemer, og mye om binær representasjon.

Vi la vekt på å beskrive algoritmer for overgang mellom tallene og deres binære representasjoner, både for heltall og desimaluttrykk.

Det gjenstår å utforme pseudokoden for algoritmen som finner binær desimal nr. k for et tall mellom 0 og 1.

Læringsmålene for dette kapitlet blir utformet senere, men de vil omfatte overganger til/fra binær representasjon både for heltall og for desimaltall.

Binære tall

Vi fortsetter med algoritmer i tilknytning til overgangen til/fra binære tall.

Den neste pseudokoden vil definere en algoritme som gir oss siffer nr. k (fra høyre) til binære-representasjonen av et tall x slik at $0 \leq x < 1$.

Hvorvidt **If-then-else**-testen er problematisk eller ikke, vil avhenge av hvordan vi representerer reelle tall som data.

Representasjon av tall som data kommer vi til i Kapittel 3.

- 1 Input x [$0 \leq x < 1$. x er det tallet vi vil finne binærformen til]
- 2 Input k [sifferet vi vil bestemme]
- 3 **For** $i = 1$ **to** k **do**

```

3.1 If  $x \geq \frac{1}{2}$  then
    3.1.1  $x \leftarrow 2(x - \frac{1}{2})$ 
    3.1.2  $z \leftarrow 1$ 
    else
    3.1.3  $x \leftarrow 2x$ 
    3.1.4  $z \leftarrow 0$ 

```

4 Output z

Binære tall

Vi illustrerer hvordan denne pseudokoden virker ved å la $x = \frac{1}{3}$ og la k være veldig stor. For de første verdiene av i skriver vi ned hva den tilhørende verdien på x vil bli, og hvilken verdi z vil ha:

1. $x = \frac{1}{3} < \frac{1}{2}$ (før) så $\frac{2}{3} = x \leftarrow 2 \cdot \frac{1}{3}$ (etter). $z = 0$.
2. $x = \frac{2}{3} \geq \frac{1}{2}$ (før) så $\frac{1}{3} = x \leftarrow 2(\frac{2}{3} - \frac{1}{2})$ (etter), $z = 1$.
3. $x = \frac{1}{3} < \frac{1}{2}$ (før) så $\frac{2}{3} = x \leftarrow 2 \cdot \frac{1}{3}$ (etter). $z = 0$.
4. $x = \frac{2}{3} \geq \frac{1}{2}$ (før) så $\frac{1}{3} = x \leftarrow 2(\frac{2}{3} - \frac{1}{2})$ (etter), $z = 1$.
-

$k = z = 0$ om k er et oddetall og $z = 1$ om k er et partall.

Aritmetikk

Vi utfører addisjon, subtraksjon, multiplikasjon og divisjon av tall på binær form omtrent som for tall i 10-tallsystemet, bortsett fra at alt i prinsippet blir mye enklere, den lille addisjonstabellen og den lille multiplikasjonstabellen blir så mye mindre.

Som eksempler regner vi følgende stykker på tavla (oppgaver for den som ikke er på forelesningen).

- $17 + 14$
- $17 - 14$
- $5 \cdot 11$
- $11 : 5$ med fire siffer bak komma.

Det er selvfølgelig mulig å finne pseudokoder som uttrykker de algoritmene vi vil bruke, men som i skolematematikken er det her best å demonstrere algoritmene ved eksempler.

OKTAL OG HEKSADESIMAL FORM

Hvis man bruker 8-tallsystemet arbeider man med tall på oktal form. Eksempelvis vil vi ha

- $443_8 = 4 \cdot 8^2 + 4 \cdot 8 + 3 = 256 + 32 + 3 = 291_{10}$

- $3,21_8 = 3 + 2 \cdot \frac{1}{8} + \frac{1}{64}$

Hvis man bruker 16-tallsystemet arbeider man med tall på heksadesimal form. Her må man supplere symbolene 1, ... 9 med sifre A, B, C, D, E og F.

Eksempelvis vil

$$2C3_{16} = 2 \cdot 16^2 + 11 \cdot 16 + 3 = 512 + 176 + 3 = 691_{10}.$$

Oktal og heksadesimal form

Fordelen med oktal og heksadesimal form er at regning med tall i disse tallsystemene representerer en rasjonalisering av regning med binære tall.

Ved å gruppere tre og tre siffer kan en binær form omgjøres direkte til oktal form:

$$101\ 100\ 001\ 010_2 = 5412_8$$

og ved å gruppere fire og fire sifre kan en binær form omgjøres til heksadesimal form:

$$1011\ 0000\ 1010_2 = B0A_{16}.$$

Oktal og heksadesimal form

Heksadesimal form brukes i 16×16 sudokuer, og til å blande farver i HTML (256 styrkegrader av hver av de tre grunnfargene), men oktale tall og heksadesimale tall ble brukt mer da man programmerte "nærmere" maskinspråket.

Vi skal ikke drille regning med tall på oktal eller heksadesimal form her.

Oktal og heksadesimal form

Oppgave (Tverrsumstest).

Gå tilbake til beviset for at tverrsumstesten for delelighet med 3 og 9 holder i 10-tallsystemet, og finn ut for hvilke tall vi har en tverrsumstest for tall på oktal og heksadesimal form.

OVER TIL KAPITTEL 3

REPRESENTASJON AV TALL I DATAMASKINER

For å forstå hvordan man lagrer tall i en datamaskin, må man vite litt om hvordan informasjon lagres generelt.

- Grunnenheten er en bit
- En *bit* vil være i en av to tilstander:
 - 0 eller 1.
 - Positiv eller negativ.
 - Sann eller usann.
 - PÅ eller AV.
 - ... eller ...
- Vi vil systematisk bruke 0 og 1 som de former for informasjon en bit kan inneholde.
- Hvordan dette skjer fysisk i hver enkelt datamaskin vil vi ikke bry oss om.

Representasjon av tall i datamaskiner

- Passe store blokker av bits kalled bytes.
- Det normale er at en byte består av 8 bits, og at vi kan skrive informasjonen i en byte som eksempelvis

10011001

- Vi kan komme til å tillate oss å arbeide med fire- bits bytes, men det er for at det da blir lettere å forklare hva som skjer.

I en byte med 8 bits kan vi lagre $2^8 = 256$ forskjellige informasjoner.
 Dette svarer til alle tall med to sifre i det heksadesimale systemet.

Representasjon av tall i datamaskiner

Hvorfor skal vi lære om hvordan tall representeres i en datamaskin?

- Vi skal se at hele tall og reelle tall er forskjellige typer tall, og at "samme" tall må representeres forskjellig når det oppfattes som heltall og når det oppfattes som reelt tall.
- Ved å kjenne til hvordan tall representeres, vil vi kjenne til begrensningene og mulige feilkilder. Hvis man skal foreta en numerisk beregning hvor antall avrundinger er i millionklassen, er det viktig å vite hvor stor feil som kan oppstå fordi representasjonen i maskinen ikke er nøyaktig. Det finnes uendelig mange tall, men bare endelig mange av dem kan representeres i en konkret maskin.
- De som utdannes fra UiO som landets fremtidige dataeksperter, bør ha generelle kunnskaper om grunnlaget for det de er eksperter på.

Representasjon av hele tall

Når vi skal representere hele tall i en datamaskin er det tre spørsmål som må besvares:

1. Hvor mange tall ønsker vi å representere?
2. Hvilke tall ønsker vi å representere?

3. Hvordan vil vi representere dem?

Svaret på spørsmål 1 er et spørsmål om hvor mange bits/ bytes vi vil sette av for å representere et enkelt tall.

Bruker vi flere bits, kan vi representere flere enkelttall, men vi vil bruke lengere tid på å manipulere tallene, og vi vil ha mindre plass til andre formål.

Representasjon av hele tall

Som eksempel bruker boka 16 bits fordelt på to bytes.

Det gir at vi kan representere $2^{16} = 65536$ forskjellige tall.

Det normale er å bruke flere bits, eksempelvis 32 bits, i ordentlige maskiner.

For lettere å vise eksempler og forklare fenomener, skal vi begrense oss til "lekemaskiner" hvor vi bruker 8 bits.

Det gir oss bare muligheten til å representere 256 tall. Prinsippene vil være de samme.

Representasjon av hele tall

Som svar på spørsmål 2, Hvilke tall ønsker vi å representere? skal vi bygge på erfaringen at det er gunstig å representere et segment av tall-linjen med omtrent like mange tall på den positive og negative siden.

Siden vi trenger en bit til å bestemme om tallet er negativt eller ikke (det vil si fortegnet), kan vi representere $2^7 = 128$ tall med fortegn + og 128 tall med fortegn -.

Det betyr at i vår lekemaskin skal vi kunne representere alle heltall a slik at

$$-128 \leq a \leq 127.$$

Representasjon av hele tall

Hvordan skal vi så representere disse tallene?

Noen valg må gjøres bare fordi vi må treffe et valg, mens andre valg peker seg umiddelbart eller etterhvert ut som de mest praktiske.

I det valget vi gjør legger vi mest vekt på følgende kriterium:

- *Vi vil kunne overføre mest mulig av aritmetikken for tall på binær form til operasjoner på representasjonene.*

Den første konsekvensen vi trekker av dette er at vi bør bruke første bit til å representere fortegnet, ettersom binær addisjon foregår fra høyre mot venstre.

Den andre konsekvensen er at vi bør la 0 i første bit representere +, og i det tilfellet la resten av de 7 bit-ene gi oss binærrepresentasjonen av tallet.

Da vil alle tallene fra 0 til 127 bli representert ved sin binære form med en ekstra ledende 0.

Representasjon av hele tall

Det kunne vært naturlig å representere et negativt tall, eksempelvis -17 som 1 og så binærformen til 17 (med 7 sifre) etterpå.

Det er to grunner til at vi vil gjøre det anderledes:

- Vi ville få to representasjoner av 0 og ingen av -128 .
- Vi ville ikke kunne brukt samme prosedyrer for addisjon og subtraksjon for positive og for negative tall.

Vi skal lage en prosedyre for å trekke fra 1, og som virker for representasjon av tall > 0 , og vi skal se på hva slags representasjoner vi da får hvis vi følger prosedyren over i de negative tallene.

Representasjon av hele tall

Eksempel (Trekk fra 1).

```
1 Input  $x_1 \cdots x_8$  [Representasjonen av tallet]
2  $i \leftarrow 8$ 
3 While  $i > 0$  do
  3.1 If  $x_i = 1$  then
    3.1.1  $x_i \leftarrow 0$ 
    3.1.2  $i \leftarrow 0$ 
  else
    3.1.3  $x_i \leftarrow 1$ 
    3.1.4  $i \leftarrow i + 1$ 
4 Output  $x_1 \cdots x_8$ 
```

Representasjon av hele tall

Eksempel (Noen inputrepresentasjoner).

- Representasjonen av 97 ($= 64 + 32 + 1$) vil være 01100001, og gir vi dette som input i pseudokoden får vi 01100000, som er representasjonen av 96 ($64 + 32$).
- Bruker vi 01100000 som input, får vi 01011111, som representerer

$$95 = 64 + 16 + 8 + 4 + 2 + 1.$$

- Bruker vi representanten for 0, 00000000, som input, får vi 11111111 som svar.
- Det kan derfor være naturlig å la 11111111 representere -1

Representasjon av hele tall

- Vi lar altså byten hvor 1 står i første bit, og binærrepresentasjonen av $127 = 128 - 1 = 128 + (-1)$ i resten av bit-ene, representere -1 .
- Bruker vi pseudokoden vår på 11111111, får vi 11111110, og det bør være representasjonen av -2 .
- Dette er det samme som 1 etterfulgt av binærrepresentasjonen av $126 = 128 - 2 = 128 + (-2)$
- Vi ser at hvis n er et tall ekte mellom 0 og 129, så peker 1 etterfulgt av binærrepresentasjonen til $128 + (-n)$ seg ut som en egnet representasjon av $-n$

Forelesning 4

Tall som data

Dag Normann - 23. januar 2008

Valg av kontaktpersoner/tillitsvalgte

- Før vi tar pause skal vi velge to til fire tillitsvalgte/kontaktpersoner.
- Kontaktpersonene skal være med på å utforme midtveisevalueringen av MAT1030.
- Vi vil også samle kontaktpersoner, foreleser, plenumsregner og/eller gruppelærerne for gjensidige tilbakemeldinger når noen ønsker det.
- Vi kan foreta valget nå eller de siste minuttene av første time.

Oppsummering av kapittel 2

I Kapittel 2 lærte vi om tall i alternative tallsystemer, i hovedsak om binære tall, oktale tall og heksadesimale tall.

Det man må kunne i tilknytning til Kapittel 2 er:

- Skrive hele tall og desimaltall på binær form.
- Finne verdien av tall skrevet på binær form.
- Utføre enkle addisjons-, subtraksjons- og multiplikasjonsstykker i binær representasjon.
- Kjenne definisjonene av oktale tall og heksadesimale tall.

Representasjon av hele tall

Mandag så vi på hvordan vi kan representere de hele tallene fra -128 til 127 ved å legge informasjon inn i en byte på 8 bits.

I boka bruker man 2 bytes og i virkeligheten gjerne 4 bytes til å representere et enkelt tall.

Prinsippene og begrunnelsen for de valg man treffer vil være de samme.

Representasjon av hele tall

Definisjon (Representasjon av hele tall).

Hvis vi har k bits til disposisjon for å representere hele tall, ($k = 8$, $k = 16$ og $k = 32$ er de mest aktuelle) representerer vi alle hele tall a slik at $-2^{k-1} \leq a < 2^{k-1}$ på følgende måte:

- Hvis $a \geq 0$, er første bit 0 og resten er det binære tallet for a med $k - 1$ sifre

- Hvis $a < 0$ er første bit 1 og resten er det binære tallet for $2^{k-1} + a$

Representasjon av hele tall

Eksempel (Noen representasjoner med $k = 8$).

La $a = -23$. Da er $a < 0$ så første bit må være 1.

$$128 + (-23) = 105 \text{ og } 1101001_2 = 105$$

Det gir at representasjonen av a er 11101001.

La $b = -99$. Første bit må være 1.

$$128 + (-99) = 29 = 11101_2$$

Vi skal bruke 7 sifre, så vi bruker at

$$11101_2 = 0011101_2.$$

Det gir representasjonen 10011101 for b .

Representasjon av hele tall

Dette synes som en tungvint måte, og det er det.

Hvis vi prøver å gjennomføre de samme eksemplene ved å regne binært, ser vi en morsom effekt:

Representasjon av hele tall

Eksempel (En alternativ måte).

La $a = -23$.

$$23 = 00010111_2$$

Vi vil regne ut $128 - 23$ binært.

$$1000000_2 - 00010111_2 = 01101001$$

så binærkoden til $128 - 23$ er 1101001.

Representasjonen av -23 blir da 11101001.

La $c = -32$

$32 = 100000_2$, og $128 - 32$ binært blir

$$10000000_2 - 100000_2 = 01100000_2.$$

Representasjonen av -32 blir da 11100000

Representasjon av hele tall

Proessen med å finne representasjonen av $-n$ fra binærformen til n kan beskrives enda enklere, ved det som boka kaller 2-er komplementet til binærformen: (Vi bruker tilnærmet samme betegnelse som boka, selv om det ikke blir god norsk)

Definisjon.

La $a_1 \cdots a_k$ være en sekvens av 0'er og 1'ere, med en 0 lengst til venstre, og med minst ett 1-tall.

2-er komplementet til sekvensen får vi ved å starte fra høyre, lese ett og ett tall og

- Alle 0'er til høyre beholdes.
- Første 1-tall fra høyre beholdes.
- Alle tall til venstre for første 1-tall endres fra 0 til 1 eller fra 1 til 0.

Representasjon av hele tall

Hvis $x_1 \cdots x_8$ er datarepresentasjonen av et positivt heltall n , er 2-er komplementet representasjonen av $-n$.

Dette gjelder selvfølgelig om vi bruker 16-bits representasjoner eller 32-bits-representasjoner også.

Representasjon av hele tall

Oppgave.

Finn en pseudokode for å beregne 2-er komplementet til en sekvens av lengde 8, av lengde 16 og av lengde 32.

Oppgave.

Hvis vi tar 2-er komplementet av 2-er komplementet av en sekvens $x_1 \cdots x_k$ av nuller og enere, får vi den opprinnelige sekvensen tilbake. Forklar hvorfor.

Representasjon av hele tall

Vi har valgt å representere hele tall som sekvenser av bits av fast lengde slik at

- De ikke-negative tallene vi representerer er representert ved sin binære form.
- Hvis vi kan representere a og b og $a + b$ finner vi representasjonen av $a + b$ ved å bruke standardalgoritmen for addisjon av binære tall på representasjonene til a og til b .
- Hvis vi kan representere a og b og $a - b$, finner vi representasjonen av $a - b$ ved å følge standardalgoritmen for subtraksjon av binære tall på representasjonene av a og b .

Representasjon av hele tall

Vi skal se på ett eksempel:

Eksempel (8 bits representasjon).

- Vi skal finne representasjonen av $23 + (-47) = -24$.
- Representasjonen av 23 er 00010111.
- $128 - 47 = 81$ og binærformen til 81 er 1010001, så -47 er representert av 11010001.
- Binærformen til 47 er 00101111, og 11010001 er også 2's komplementet til 00101111.
- Legger vi sammen 00010111 og 11010001 som binære tall får vi 11101000.
- $1101000_2 = 64 + 32 + 8 = 104$
- Det betyr at 11101000 er representasjonen av $-(128 - 104) = -24$.

Representasjon av hele tall

Hva skjer hvis vi regner oss ut over de tallene vi har representert?

-128 har representasjon 10000000.

Dette er også binærtallet til 128.

Hvis vi legger 1 til 127 via representasjonen 01111111, får vi representasjonen 10000000 for -128 .

I en viss forstand vil representasjonene identifisere 128 og -128 , og tall-linjen erstattes av en tall-sirkel hvor vi regner rundt og rundt.

Disse betraktningene har liten direkte relevans for informatikk, og vi skal ikke forfølge dem videre.

Representasjon av reelle tall

Når vi skal representere reelle tall i en datamaskin har vi andre hensyn å ta enn når vi representerer hele tall.

Vi ønsker å representere tall slik at vi kan bruke maskinene til å regne på

- Avstander i verdensrommet.
- Vekt av himmellegemer (med beregning av gravitasjonskrefter).
- Værdata.
- Strekk- og brekkbelastninger på brokonstruksjoner.
- Molekyl- og atomvekter.
- Konstanter i forbindelse med massevirkningsloven.
- Mye, mye mer.

Representasjon av reelle tall

- Siden vi bare kan representere endelig mange reelle tall i en maskin, må vi regne med avrundede størrelser.
- I endel utregninger, eksempelvis under utarbeidelse av værprognoser, må vi gjenta enkelte operasjoner mange, mange ganger, og avrundingsfeilene kan vokse.
- Vi trenger derfor meget stor presisjon på de tallene vi kan representere, slik at det er god nok presisjon selv etter mange avrundinger.

Representasjon av reelle tall

Vi ønsker datamaskiner som arbeider fort og uten å bruke for mye plass, men som samtidig kan brukes til et vidt spekter av regneoppgaver.

I valg av hvordan vi representerer reelle tall i datamaskiner, må vi ta hensyn til følgende:

- Vi skal kunne representere både de positive og negative tallene og både de svært store (små) tallene og tall svært nært 0.
- Vi vil ha høy presisjon på de tallene vi representerer.
- Det skal finnes effektive algoritmer som "oversetter" de vanlige aritmetiske operasjonene til representasjoner.

Representasjon av reelle tall

For å forstå prinsippene for representasjon av tall i en maskin, kan vi se på hvordan reelle tall fremstilles på en lommeregner eller i en tekst som omhandler store eller små tall. I stedetfor å skrive

23085010000000000000

kan vi skrive

$0,2308501 \cdot 10^{21}$

eller

2,308501E20

Med unntak av at vi vil bruke binære tall i stedetfor vanlige tall fra titallsystemet, vil vi kode reelle tall via tre informasjonsbiter, fortegn, sifrene brukt og eksponent.

Representasjon av reelle tall

Vi vil skrive tallene på normalisert binær form:

- Et fortegn, + eller –.
- Et binært desimaluttrykk på formen

$0,1\dots$

kalt signifikanden.

- En eksponentdel

2^e

hvor e er et heltall.

Representasjon av reelle tall

Eksempel.

a) $0,101100101 \cdot 2^{-3}$ er på normalisert binærform.

Vi kunne skrevet dette som $0,000101100101$.

b) $101,0101101$ er ikke på normalisert binærform.

Da burde vi skrevet $0,1010101101 \cdot 2^3$

c) $0,1000010100001 \cdot 2^{28}$ er på normalisert binær form.

I dette tilfellet er det liten grunn til å skrive tallet på eksakt form, og det ville også antydnet en større nøyaktighet enn det vi trolig har grunnlag for.

d) $0,11010 \cdot 2^{-87}$ er på normalisert binær form.

Representasjon av reelle tall

Vi har prosedyrer for å utføre aritmetikk på tall på normalisert binær form:

- Addisjon
Om nødvendig, flytt komma i det ene tallet slik at vi får samme eksponent.
Legg sammen signifikandene
Normaliser resultatet om nødvendig.
- Multiplikasjon
Multipliser signifikandene
Adder eksponentene
Normaliser om nødvendig.

Representasjon av reelle tall

Det finnes selvfølgelig tilsvarende prosedyrer for subtraksjon og divisjon.

Normalisert representasjon er like godt egnet til multiplikasjon og divisjon som til addisjon og subtraksjon.

Representasjon av reelle tall

Hvis man vil danne seg et inntrykk av hva disse binære eksponentene betyr i 10-tallsystemet, bruker vi formelen ert

$$2^k = 10^{k \cdot \log 2}$$

og minner om at $\log 2 = 0,301$ sånn omtrent.

Representasjon av reelle tall

Vi vil følge boka i beskrivelsen av hvordan reelle tall representeres på en datamaskin. Et interessant aspekt er at vi ikke vil representere tallet 0 på maskinen.

- Vi bruker 32 bits til å representere ett tall.
- Det første bit'et bruker vi til å representere fortegnet.
- De neste 8 bit'ene bruker vi til å representere eksponenten.
- De siste 23 bit'ene bruker vi til å representere signifikanden.

Representasjon av reelle tall

- For fortegnet bruker vi 0 for + og 1 for −.
- For signifikanden bruker vi det 23-sifrede binære tallet som står bak komma. Dette svarer til mellom 7 og 8 sifre i ti-tallsystemet, så det er den nøyaktigheten vi regner med. Avrundingsfeil vil da være i størrelsesorden 2^{-23} . Siden maskinen ofte bare vil ignorere sifre for langt ute, og ikke forhøye på vanlig måte, blir avrundingsfeilen større enn man først kunne tro.
- Man kan bruke dobbel presisjon, det vil si 64 bits, til å representere et tall, hvis det er fare for for mange avrundingsfeil. Vi skal ikke se på detaljene her.

Representasjon av reelle tall

- For å representere eksponenten, bruker vi 8 bits. Det gir oss mulighet til å fange opp $2^8 = 256$ forskjellige eksponenter.
- Vi har bruk for å representere omtrent like mange negative som positive eksponenter.

Representasjon av reelle tall

Vi kunne brukt representasjonen vår av alle tall fra -128 til 127 som vi beskrev for "lekedatamaskinen", og det kunne endog vært en teknologisk sett bedre løsning, ettersom vi har bruk for å addere og subtrahere eksponenter.

Standard metode er imidlertid at man representerer en eksponent ved summen av binærformen og 01111111_2 , hvor $01111111_2 = 127$

Dette plasserer representasjonen av 0 omtrent midt i, og gir oss mulighet for å representere alle eksponenter fra -127 til 128 .

Representasjon av reelle tall

Eksempel.

Vi vil finne datarepresentasjonen av tallet $2,5_{10}$

Først må vi skrive tallet på binær form:

$$2,5_{10} = 10,1_2.$$

Den normaliserte binære formen er

$$0,101 \cdot 2^2.$$

Representasjon av reelle tall

Eksempel (fortsatt).

Da vil vi bruke 0 for å representere fortegnet, 10000001 for å representere eksponenten og 1010000000000000000000 til å representere signifikanden.

$2,5_{10}$ representeres da av bitsekvensen

01000000 11010000 00000000 00000000

fordelt på 4 bytes.

Representasjon av reelle tall

Eksempel.

Finn representasjonen av

$$-\frac{1}{3}.$$

Binærformen med 23 gjeldende siffer er

$$\left(-\frac{1}{3}\right)_{10} = -0,0101010101010101010101_2.$$

Normalisert binær form blir da

$$-0,10101010101010101010101 \cdot 2^{-1}.$$

Representasjon av reelle tall

Eksempel (fortsatt).

Da må vi bruke

- 1 for å representere fortegnet.
- 01111110 for å representere eksponenten.
- 10101010101010101010101 for å representere signifikanden.

Representasjonen, fordelt på 8 bytes, blir da

10111111 01010101 01010101 01010101.

Representasjon av reelle tall

- De "reelle tallene" vi regner med i en datamaskin, kalles ofte flytende binære punkter.
- Det er vanlig å regne med slike flytende punkter med enkel eller dobbel presisjon.
- Ulempen er at man ikke har god kontroll på feilene som gjøres, det vil si, man vet ofte ikke hva usikkerheten i resultatet er.
- Det arbeides, på forskningsnivå, med andre tilnærminger til beregninger over reelle tall hvor man har bedre kontroll på usikkerhetene i svarene.

Generelt om representasjoner

Anta at D_1 og D_2 er to datatyper i "den virkelige verden".

Vi ønsker å finne effektive programmer for visse operasjoner som sender data av type D_1 til data av type D_2

$$\begin{array}{ccc} D_1 & \rightarrow \text{operasjon} \rightarrow & D_2 \\ \downarrow \uparrow & & \downarrow \uparrow \\ \text{Rep} & & \text{Rep} \\ \downarrow \uparrow & & \downarrow \uparrow \\ R_1 & \rightarrow \text{løfting} \rightarrow & R_2 \end{array}$$

Det er viktig at løftingen kan utføres raskt og uten bruk av for mye ressurser.

Det er ikke alltid slik at det bare er en måte å representere et objekt på, men en representant kan bare representere ett objekt.

OVER TIL KAPITTEL 4

LOGIKK

Kapittel 4 i læreboka gir en kort innføring i viktige deler av logikken.

Hvorfor skal informatikkstudenter lære noe om logikk?

- von Neumanns konstruksjon av datamaskinen er basert på utsagnslogikk og logiske porter.
- Grunnarkitekturen av datamaskiner har ikke endret seg mye siden den tid, selv om de teknologiske forbedringene har vært enorme.
- Vi bruker språket fra logikk til å formulere forutsetninger P i kontrollstrukturer som

If P then ... else ...

- Vi bruker logikk for automatisk å sikre at forskjellige data lagt inn i en base er forenlige med hverandre.
- Vi trenger logikk for å kunne definere hva som menes med korrekthet av et program.
- Kunstig intelligens, sikkerhetsprotokoller for utveksling av informasjon og mye annen avansert bruk av datamaskiner bygger på logikk.

Logikk

- Logikk, slik vi kjenner faget i dag, har sin opprinnelse fra gresk filosofi og vitenskap.
- Tanken var at et resonement må kunne stykkes opp i enkelte tankeprang, grunnresonement, hvor det vil være lett å bestemme om grunnresonementene er riktige eller representerer feilslutninger.
- Da kan man finne ut av hvilke deler av et argument som baserer seg på ren tankekraft, og hva som baserer seg på unevnte forutsetninger.
- Aristoteles formulerte en del syllogismer som skulle gi oss de lovlige logiske slutningene.
- Vi skal ikke gå inn på den klassiske syllogismelæren, men etterhvert legge grunnlaget for den.

Logikk

Eksempel.

- a) – Jeg rekker ikke middagen.
- Du kommer senere hjem enn meg.
- Altså rekker ikke du middagen.

- b) – Jeg rekker ikke middagen.
- Hvis jeg ikke rekker middagen, rekker ikke du middagen.
- Altså rekker ikke du middagen.

I eksempel a) har vi en skjult forutsetning, mens i eksempel b) er argumentet logisk sett riktig. Vi kan skrive dette argumentet om til et annet, hvor b) holder, men a) ikke holder.

Logikk

Eksempel.

- a) – Jeg liker ikke Bamsemums
- Du liker alt jeg liker
- Altså liker ikke du Bamsemums
- b) – Jeg liker ikke Bamsemums
- Hvis jeg ikke liker Bamsemums, liker ikke du Bamsemums
- Altså liker ikke du Bamsemums.

Her er det opplagt en feil i argument a), mens b) holder fortsatt.

Logikk

Som en tommelfingerregel kan vi si at

Et argument er logisk holdbart hvis vi kan bytte ut delformuleringer som ikke inneholder noe av den logiske strukturen med andre formuleringer uten at argumentet blir feil.

Hovedutfordringen blir å bestemme hva som tilhører den logiske strukturen og hva vi kan forandre på for å bruke testen over.

Logikk

Før vi starter på fagstoffet skal vi trekke frem fire navn som er viktige for utviklingen av logikk i det 20. århundre og for sammenfiltringen av matematisk logikk og teoretisk informatikk:

- Toralf Skolem
- Kurt Gödel
- Alan Turing
- John von Neumann

Forelesning 5

Logikk

Dag Normann - 28. januar 2008

Oppsummering av Kapittel 3

- I Kapittel 3 så vi på hvordan data, som hele tall og reelle tall, kan representeres som bit-sekvenser i en datamaskin.
- Stoffet er til dels sammenfallende med deler av MAT-INF1040, men ikke nok til å gi studiepoengreduksjon.
- Læringsmålene i stoff fra Kapittel 3 kan sammenfattes som:
 - Man skal vite prinsippene for å representere hele tall og reelle tall som bit-sekvenser i en datamaskin, og skal kunne begrunne en del av de valg som er gjort i måten dette blir gjort på.
 - Gitt rammene for hvor mange bits som brukes til de enkelte formål, skal man kunne finne representasjonen av et tall i konkrete tilfeller.

LOGIKK

Forrige onsdag begynte vi på Kapittel 4 om logikk.

Vi snakket en del om hvorfor informatikkstudenter bør lære seg noe logikk, og litt om hvordan logikk brukes i teknologiske anvendelser.

Illustrert med et par eksempler om middag og Bamsemums diskuterte vi logikkens rolle i studiet av hva som er et logisk holdbart resonnement (se 23.01.2008).

Logikk

Hvis en datamaskin skal kunne sjekke gyldigheten av et resonnement, må vi laste ned alle skjulte forutsetninger i resonnementet.

Vi må også laste ned hvilke atomære resonnementer som er lovlige, for en maskin kan bare kontrollere om noe er utført i tråd med forhåndsbestemte regler.

Hvis en maskin skal kunne "forstå" hva som tilhører den logiske strukturen i en formulering, må den knyttes til bruk av spesielle tegn eller ordsekvenser.

Dette er helt analogt med den rigiditeten som kreves av et program i et programmeringsspråk.

Hva skal vi lære av logikk?

- Utsagnslogikk
- Predikatlogikk
- Litt om hvordan man fører et bevis

- Algoritmer for å teste om utsagn er logisk holdbare eller ikke

Utsagnslogikk

Definisjon.

Et utsagn er en ytring som enten er sann eller usann.

- Som matematisk definisjon er ikke denne definisjonen spesielt god, ettersom den ikke kan brukes til å bestemme hva som er utsagn og hva som ikke er det.
- Er "Per er en dannet mann" et utsagn?
- Vi vil betrakte dette som et utsagn, ettersom ytringen i en gitt situasjon uttrykker en oppfatning som enten kan aksepteres eller bestrides.
- Mange filosofer vil være uenige med oss her.

Utsagnslogikk

Eksempel.

Følgende er eksempler på utsagn, slik vi skal bruke begrepet:

- $2^{10} < 1000$
- $\pi \neq e$
- Anne har røde sko.
- I morgen blir det pent vær.
- Det finnes mange grader av uendelighet.

Utsagnslogikk

Eksempel.

Følgende ytringer kan ikke oppfattes som utsagn:

- Når går toget?
- Uff!!!
- Dra til deg den lurvete mærschedesen din, eller så kjører jeg på den! (Sitat fra sint trikkefører i Grensen.)
- Måtte sneen ligge lenge og løypene holde seg.

Utsagnslogikk

Eksempel.

Vi har sett endel utsagn i forbindelse med formuleringer av kontrollstrukturer i pseudokoder:

- **While** $i > 0$ **do**
- **Repeat** ... **until** $x > k$
- **If** x partall **then** ... **else** ...

Under en utregning vil verdiene på variablene endre seg, men ved hvert enkelt regneskritt vil "ytringene" enten være sanne eller usanne, og vi ser derfor på dem som utsagn.

Utsagnslogikk

Eksempelene på forrige side aktualiserer spørsmålet om matematiske likninger, ulikheter og andre formler hvor det forekommer variable størrelser kan betraktes som utsagn:

- $x^2 + 2x - 1 = 0$
- $\sin^2 x + \cos^2 x = 1$
- $f(x) = f'(x)$

Det første tilfellet er en likning i variabelen x , det andre en kjent identitet fra trigonometrien og det siste en differensiallikning hvor f er den ukjente.

For at vi skal slippe å slåss om dette er eksempler på utsagn eller ikke, innfører vi et nytt begrep, et predikat.

Utsagnslogikk

Definisjon.

Et predikat er en ytring som inneholder en eller flere variable, men som vil bli sant eller usant når vi bestemmer hvilke verdier de variable skal ha.

Eksempel.

Alle eksemplene fra forrige side, $x^2 + 2x - 1 = 0$, $\sin^2 x + \cos^2 x = 1$ og $f(x) = f'(x)$, er eksempler på predikater.

I de to første tilfellene er x variabelen, og i det siste tilfellet er både f og x variable.

Utsagnsvariable og sannhetsverdier

Det er ikke så viktig å vite hva et utsagn er. Det viktige er at når vi betrakter en ytring som et utsagn, stripper vi ytringen for alt untatt egenskapen at den vil være sann eller usann.

Vi vil bruke bokstaver p , q , r og liknende som utsagnsvariable, det vil si at de kan stå for et hvilket som helst utsagn.

Vi vil la T og F stå for de to sannhetsverdiene sann og usann (*true* og *false*).

Hver utsagnsvariabel p kan da ha en av verdiene T eller F .

Utsagnsvariable og sannhetsverdier

Det finnes mange andre valg av bokstaver eller symboler for å betegne de to sannhetsverdiene sann og usann på.

Noen eksempler er

- \top og \perp
- 1 og 0.
- **S** og **G** (i norske fremstillinger)

Utsagnslogiske bindeord, konnektiver

Eksempel.

La oss se på følgende prosedyre:

- 1 Input x [$x \geq 0$, x heltall]
- 2 Input y [$y \geq 0$, y heltall]
- 3 **While** $x > 0$ og $y > 0$ **do**
 - 3.1 $x \leftarrow x - 1$
 - 3.2 $y \leftarrow y - 1$
- 4 $z \leftarrow x + y$
- 5 Output z

Dette er en algoritme for å beregne $|x - y|$ fra x og y .

Utsagnslogiske bindeord, konnektiver

Definisjon.

- Hvis p og q er to utsagn, er uttrykket $p \wedge q$ også et utsagn.
- Vi leser det p **og** q .
- $p \wedge q$ er sann hvis både p og q er sanne, ellers er $p \wedge q$ usann.
- Vi kaller ofte $p \wedge q$ for konjunksjonen av p og q .

Utsagnslogiske bindeord, konnektiver

Definisjonen av når $p \wedge q$ er sann kan gis i form av en tabell.

En slik tabell kaller vi en sannhetsverditabell.

Utarbeidelsen av sannhetsverditabeller vil være en viktig ferdighet i dette kurset:

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

Utsagnslogiske bindeord, konnektiver

I en matematikk/informatikksammenheng er det greit å bruke symbolet \wedge :

- $3 \leq x \wedge x \leq 5$ er en helt grei formulering.
- **While** $x > 0 \wedge y > 0$ **do** kan være en alternativ måte å starte while-løkka fra eksemplet vårt på.
- Ofte vil man finne at man bruker samme typografi som for andre kontrollstrukturer i denne sammenhengen:
While $x > 0$ **and** $y > 0$ **do**

Utsagnslogiske bindeord, konnektiver

Hvis man gjengir sammensetning av utsagn i dagligtale, er det bedre å bruke ordet "og".

Man må imidlertid være klar over at den utsagnslogiske forståelsen visker ut noen av de nyansene vi kan legge inn i dagligtale.

I de to første eksemplene på neste side vil utsagnslogikken fange opp meningen, mens vi i de to neste mister mye av meningen.

Utsagnslogiske bindeord, konnektiver

Eksempel.

1. Per er født i Oslo og Kari er født i Drammen
2. Jeg liker å spille fotball og jeg liker å drive med fluefiske.
3. Jeg gikk inn i stua og tok av meg skiene/Jeg tok av meg skiene og gikk inn i stua.
4. Jeg bestilte snegler til forrett og du forlot meg rasende/Du forlot meg rasende og jeg bestilte snegler til forrett.

Utsagnslogiske bindeord, konnektiver

Det neste bindeordet vi skal se på er 'eller'.

Dette ordet kan ha to betydninger, og vi må velge en av dem.
Dette kommer vi tilbake til etter et par eksempler.

Utsagnslogiske bindeord, konnektiver

Eksempel.

La oss ta utgangspunkt i følgende kontrollstruktur:

- 1 Input x [$x \geq 0$, x heltall]
 - 2 Input y [$y \geq 0$, y heltall]
 - 3 $z \leftarrow 0$
 - 4 **While** $x > 0$ eller $y > 0$ **do**
 - 4.1 $x \leftarrow x - 1$
 - 4.2 $y \leftarrow y - 1$
 - 4.3 $z \leftarrow z + 1$
 - 5 Output z
- Dette gir oss en algoritme for å beregne $\max\{x, y\}$.

Utsagnslogiske bindeord, konnektiver

Hvis vi gir x og y store verdier n og m , vil både x og y ha positive verdier etter noen få gangers tur i while-løkken, vi ønsker ikke at løkken skal stoppe av den grunn.

Derfor vil vi gjerne at et utsagn "p eller q" skal kunne være sant også når både p og q er sanne, i det minste i denne sammenhengen.

Er $2 \leq 3$? Er $3 \leq 3$?

I en matematisk sammenheng vil vi gjerne at begge deler skal være sanne, vil jo at $x \leq 3$ skal være oppfylt både av de tallene som er ekte mindre enn 3 og av 3 selv. Det betyr at når et av leddene i et eller-utsagn er sant, vil vi at hele utsagnet skal være sant.

$$x \leq y \text{ er det samme som } x < y \vee x = y$$

Utsagnslogiske bindeord, konnektiver

Definisjon.

- Hvis p og q er to utsagn, er uttrykket $p \vee q$ også et utsagn.
- Vi leser det p **eller** q.
- $p \vee q$ er sann hvis p, q eller begge to er sanne, ellers er $p \vee q$ usann.
- Vi kaller $p \vee q$ for disjunksjonen av p og q.

Utsagnslogiske bindeord, konnektiver

Definisjonen av når $p \vee q$ er sann kan også gis i form av en sannhetsverditabell:

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Utsagnslogiske bindeord, konnektiver

Eksempel.

Følgende eksempler fra dagligtale viser at det er to forskjellige måter å forstå ordet 'eller' på

- Du kan få servere pølser eller du kan få servere pizza i bursdagsselskapet.
- Vil du ha en PC eller vil du ha en Mac?
- Jeg kommer til middag om toget er i rute eller om jeg får sitte på med en kollega.
- Om du leser VG eller om du leser Dagbladet finner du ikke noe stoff om hyperbolsk geometri.

Utsagnslogiske bindeord, konnektiver

Vi skal bruke den inklusive betydningen av *eller*, og vi bruker symbolet \vee eller kontrollstrukturvarianten **or**.

Vi vil bruke dette bindeordet i en matematikk/informatikksammenheng, og være var-somme med å overføre den inklusive tolkningen til dagligtale.

Ekklusiv *eller* kan også defineres ved en sannhetsverditabell.

Dere utfordres til å gjøre dette selv.

Utsagnslogiske bindeord, konnektiver

I enkelte programmeringssammenhenger, trenger vi å nyansere forståelsen av \vee og av \wedge ytterligere.

Anta at $P(\vec{x})$ og $Q(\vec{x})$ er to prosedyrer (\vec{x} er en vanlig måte å skrive en generell sekvens av variable på), slik at vi ikke kan være sikre på om de tilhørende programmene alltid terminerer.

Anta at vi bruker et programmeringsspråk som tillater kontrollstrukturer av tilnærmet form

If $P(\vec{x}) > 0$ **or** $Q(\vec{x}) > 0$ **then** ...

Skal vi da kunne fortsette når $P(\vec{x})$ ikke har noen verdi, men $Q(\vec{x}) > 0$?

Diskusjonen foregår muntlig på forelesningen.

Utsagnslogiske bindeord, konnektiver

Det neste ordet vi skal se på er ikke i betydningen *det er ikke slik at*

Utsagnslogiske bindeord, konnektiver

Eksempel.

- Månen er ikke full i morgen.
- Hurtigruta går ikke innom Narvik.
- Jeg rekker ikke middagen
- Jeg liker ikke Bamsemums.

I alle disse tilfellene benekter vi en positiv påstand, eksempelvis "Jeg liker Bamsemums".

Utsagnslogiske bindeord, konnektiver

Eksempel.

- 1 Input x [$x \geq 0$ heltall]
- 2 Input y [y heltall, $0 \leq y \leq x$]
- 3 **While** $y \neq 0$ **do**
 - 3.1 $z \leftarrow y$
 - 3.2 $y \leftarrow \text{rest}(x, y)$ [$\text{rest}(x, y)$ gir restdelen når x deles på y]
 - 3.3 $x \leftarrow z$
- 4 Output x

Dette er en måte å formulere Euklids algoritme på (en måte å finne største felles faktor i to tall på).

Poenget her er formuleringen $y \neq 0$, en benektelse av at $y = 0$.

Utsagnslogiske bindeord, konnektiver

Vi vil vil bruke et spesielt tegn, \neg for å uttrykke at vi benekter et utsagn.

Definisjon.

- Hvis p er et utsagn, er $\neg p$ et utsagn.
- $\neg p$ får sannhetsverdien F om p har sannhetsverdien T og $\neg p$ får sannhetsverdien T om p har sannhetsverdien F.
- Vi kaller $\neg p$ for negasjonen av p .

Utsagnslogiske bindeord, konnektiver

Vi kan også gi denne definisjonen på sannhetsverditabellform:

p	¬p
T	F
F	T

Denne tabellen er selvforklarende.

Sammensatte utsagn

Vi skal snu litt på rekkefølgen av stoffet i forhold til læreboka.

Ved å bruke konnektivene \wedge , \vee og \neg har vi gitt utsagnslogikken sin fulle uttrykkskraft.

De konnektivene vi skal se på senere, kan erstattes med sammensatte uttrykk hvor vi bare bruker \neg , \wedge og \vee .

Det er faktisk mulig å klare seg med bare \neg og \wedge eller bare med \neg og \vee , men da trenger vi sammensatte utsagn som det er vanskelig å lese.

For å fortsette denne diskusjonen, må vi se på hva vi mener med sammensatte utsagn.

Sammensatte utsagn, bruk av parenteser

Vi har sett at $x \neq 0$ egentlig er en alternativ skrivemåte for $\neg(x = 0)$.

Anta at vi i en programmeringssammenheng har bruk for å uttrykke betingelsen $x \neq 0$ og $y > 0$.

Dette burde vi kunne skrive som

$$\neg(x = 0) \wedge y > 0.$$

Hvis p er utsagnet $x = 0$, q er utsagnet $y > 0$ og r er utsagnet $p \wedge q$, skal $\neg r$ være utsagnet $\neg p \wedge q$?

Det var vel ikke det vi mente, \dots , eller?

Sammensatte utsagn, bruk av parenteser

Vi vil bruke parenteser for å markere rekkevidden av et konnektiv, det vi si,

hva vi mener med p og q når vi skriver $\neg p$, $p \wedge q$ eller $p \vee q$.

Vi skal gi en mer formell beskrivelse av hvordan vi skal bruke parenteser senere, men praksis fra skolealgebraen vil være rettningsgivende.

I eksemplet fra forrige side, kan vi skrive

$$\neg(x = 0 \wedge y > 0)$$

hvis vi mener å negere hele konjunksjonen, mens vi vil skrive $\neg(x = 0) \wedge y > 0$ hvis det bare er $x = 0$ som skal benektes.

Sammensatte utsagn, bruk av parenteser

For tydeligere å se forskjellen, kan vi regne ut sannhetsverditabellen til de to sammensatte uttrykkene $\neg p \wedge q$ og $\neg(p \wedge q)$.

En sannhetsverditabell for et sammensatt uttrykk vil være en tabell hvor vi starter med

- En kolonne for hver utsagnsvariabel.
- En kolonne for hver del av det gitte utsagnet.
- En rad for hver mulig fordeling av sannhetsverdier på utsagnsvariablene
- For hvert delutsagn skriver vi den sannhetsverdien delutsagnet vil ha i hver rad ut fra hvilke sannhetsverdier utsagnsvariablene har.

Sammensatte utsagn, bruk av parenteser

Eksempel ($\neg(p \wedge q)$).

p	q	$p \wedge q$	$\neg(p \wedge q)$
T	T	T	F
T	F	F	T
F	T	F	T
F	F	F	T

Sammensatte utsagn, bruk av parenteser

Eksempel ($\neg p \wedge q$).

p	q	$\neg p$	$\neg p \wedge q$
T	T	F	F
T	F	F	F
F	T	T	T
F	F	T	F

Sammensatte utsagn, bruk av parenteser

Det er ikke noe i veien for å lage sannhetsverditabeller for sammensatte utsagn med tre eller flere utsagnsvariable.

I det neste eksemplet skal vi se på et sammensatt utsagn $(p \wedge q) \vee (\neg p \wedge r)$.

Det finnes 8 forskjellige måter å fordele sannhetsverdiene til tre variable på.

Det betyr at tabellen vår må ha 8 linjer under streken.

Med fire variable får vi 16 linjer.

Det vil ikke få plass på skjermen, så da må vi utvikle andre metoder.

Tabellen overføres til onsdagens manuskript.

Forelesning 6

Logikk

Dag Normann - 30. januar 2008

Sammensatte utsagn, sannhetsverditabeller

Mandag 28/1 innførte vi bindeordene (konnektivene) \wedge for **og**, \vee for **eller** og \neg for **ikke**. Vi så hvordan vi kunne definere disse tre konnektivene ved hjelp av sannhetsverditabeller.

Vi drøftet litt om hvordan man bør bruke parenteser.

Rekkevidden til \neg er det neste fulle utsagnslogiske uttrykket, mens rekkevidden til \wedge og \vee vil "gå forbi" forekomster av \neg .

Sammensatte utsagn, sannhetsverditabeller

Vi så på noen eksempler på hvordan man kan utarbeide en sannhetsverditabell for et sammensatt uttrykk.

Vi var midt i et eksempel, som vi tar opp igjen her, da tiden var ute.

Sammensatte utsagn, sannhetsverditabeller

Eksempel $((p \wedge q) \vee (\neg p \wedge r))$.

p	q	r	$\neg p$	$p \wedge q$	$\neg p \wedge r$	$(p \wedge q) \vee (\neg p \wedge r)$
T	T	T	F	T	F	T
T	T	F	F	T	F	T
T	F	T	F	F	F	F
T	F	F	F	F	F	F
F	T	T	T	F	T	T
F	T	F	T	F	F	F
F	F	T	T	F	T	T
F	F	F	T	F	F	F

"Hvis-så" og "hvis og bare hvis"

Eksempel.

Hvis $x < 3$ så er $x < 5$.

Man skal ikke kunne mye matematikk for å mene at dette må være riktig.

Utsagnet $x < 3 \vee \neg(x < 5)$ vil anta forskjellige sannhetsverdier avhengig av hva x er.

Er det mulig å definere et utsagnslogisk bindeord \rightarrow slik at

- Hvis p og q er utsagn, så vil $p \rightarrow q$ være et utsagn slik at sannhetsverdien til $p \rightarrow q$ avhenger av sannhetsverdiene til p og til q .
- Når x varierer, skal alltid $x < 3 \rightarrow x < 5$ være sant?

“Hvis-så” og “hvis og bare hvis”

Eksempel (Fortsatt).

La $p(x)$ stå for $x < 3$ og la $q(x)$ stå for $x < 5$.

Hvis $x = 2$ vil både $p(x)$ og $q(x)$ få verdien T, så $p \rightarrow q$ bør være sant når både p og q er sanne.

Hvis $x = 4$ vil $p(x)$ få verdien F, mens $q(x)$ får verdien T. Derfor bør $p \rightarrow q$ bli sann hvis p er usann mens q er sann.

Hvis $x = 6$ blir både $p(x)$ og $q(x)$ usanne, så $p \rightarrow q$ bør bli sann også når både p og q er usanne.

“Hvis-så” og “hvis og bare hvis”

Eksempel.

Hvis $x^2 > 0$ så er $x > 0$.

Mange vil protestere på dette!

Hvorfor?

Fordi det finnes moteksempler, eksempelvis $x = -1$

Et moteksempel er et eksempel på at en ytring ikke alltid er riktig.

Et moteksempel til et utsagn “Hvis p så q ” vil alltid være et tilfelle hvor p er sann, mens q er usann.

Det vil derfor være naturlig å la $p \rightarrow q$ være usann når p er sann og q er usann.

“Hvis-så” og “hvis og bare hvis”

Definisjon.

- Hvis p og q er to utsagn, er $p \rightarrow q$ også et utsagn.
- $p \rightarrow q$ blir sann hvis q er sann eller hvis p er usann.

- Hvis p er sann og q er usann, lar vi $p \rightarrow q$ bli usann.
- Vi vil lese "hvis p så q ".

"Hvis-så" og "hvis og bare hvis"

Vi kan definere \rightarrow ved hjelp av følgende sannhetsverditabell.

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

"Hvis-så" og "hvis og bare hvis"

De som har problemer med at vi her sier at det er sant at noe usant medfører noe sant (eller noe usant) får hente støtte i det kjente sitatet fra Ibsen

hvor Utgangspunktet er galest, blir tidt Resultatet originalest.

Ibsen fanger her inn essensen av vår definisjon av hvordan sannhetsverdien til

Utgangspunkt \rightarrow Resultat

bestemmes av sannhetsverdien til utgangspunktet og til resultatet, og når utgangspunktet er noe som ikke er sant kan resultatet bli hva som helst.

"Hvis-så" og "hvis og bare hvis"

Det er ikke så naturlig å bruke \rightarrow i programmeringssammenheng. Derfor gir vi ingen eksempler hvor \rightarrow brukes i kontrollstrukturer.

Læreboka bruker ordet *implies* i forbindelse med \rightarrow .

Dette er uheldig, fordi det lett fører til en sammenblanding av symbolene \rightarrow og \Rightarrow .

$x < 5 \Rightarrow x < 3$ er regelrett feil, mens $x < 5 \rightarrow x < 3$ er sant for noen verdier av x og usant for andre.

Dette ble utdypet mere på forelesningen.

"Hvis-så" og "hvis og bare hvis"

Oppgave.

a) Finn sannhetsverditabellen til

$$(p \rightarrow q) \rightarrow p$$

b) Finn sannhetsverditabellen til

$$(p \rightarrow q) \vee (q \rightarrow p)$$

c) Hva ser du i kolonnen lengst til høyre?

“Hvis-så” og “hvis og bare hvis”

Når vi bruker “hvis-så” i dagligtale, kan vi få noe meningsløst ut av det.

I de eksemplene som følger kan vi diskutere om tolkningen som logikken forteller oss er riktig stemmer overens med den tolkningen vi vil legge i ytringen som vanlig kommuniserende mennesker:

“Hvis-så” og “hvis og bare hvis”

Eksempel.

- Hvis Noah hadde lært dyrene å svømme, ville jorda vært overbefolket av løver.
- Hvis ulven spiser Rødhette, vil det bli en Grimm historie.
- Du får gå på kino hvis du vasker opp etter maten.
- Hvis dere avholder reelle demokratiske valg, vil vi gi støtte til oppbyggingen av infrastrukturen.

“Hvis-så” og “hvis og bare hvis”

I det nest siste eksemplet gis det ikke rom for å få gå på kino hvis oppvasken ikke taes og i det siste eksemplet er tilbudet om økonomisk støtte helt klart knyttet til kravet om demoktati.

Oppvask vil være både en nødvendig og tilstrekkelig betingelse for kinobesøk.

Vi innfører et siste konnektiv, \leftrightarrow som skal fange opp hvis og bare hvis i samme forstand som \rightarrow fanger opp hvis - så-.

“Hvis-så” og “hvis og bare hvis”

Definisjon.

- Hvis p og q er utsagn, er $p \leftrightarrow q$ et utsagn.
- $p \leftrightarrow q$ er sant når både p og q er sanne, og når både p og q er usanne.
- $p \leftrightarrow q$ er sant når p og q har den samme sannhetsverdien.

“Hvis-så” og “hvis og bare hvis”

Vi kan selvfølgelig også definere \leftrightarrow ved en sannhetsverditabell:

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

“Hvis-så” og “hvis og bare hvis”

Legg merke til at om vi skriver ut sannhetsverditabellene for

- $p \leftrightarrow q$
- $(p \wedge q) \vee (\neg p \wedge \neg q)$
- $(p \rightarrow q) \wedge (q \rightarrow p)$

får vi den samme søylen lengst til høyre.

Det er en passende treningsoppgave å skrive ut de to siste tabellene.

“Hvis-så” og “hvis og bare hvis”

Eksempel ($p \rightarrow (\neg p \rightarrow q)$).

p	q	$\neg p$	$\neg p \rightarrow q$	$p \rightarrow (\neg p \rightarrow q)$
T	T	F	T	T
T	F	F	T	T
F	T	T	T	T
F	F	T	F	T

“Hvis-så” og “hvis og bare hvis”

Eksempel.

De to neste eksemplene blir bare gjennomgått på tavla:

- $p \rightarrow (q \rightarrow p)$
- $p \wedge (p \rightarrow q) \rightarrow q$

“En digresjon”

Hvis vi ønsker å være helt formelle, kan vi definere formelle utsagnslogiske uttrykk på følgende måte, hvor vi skiller mellom variable for grunnutsagn og sammensatte utsagn:

- Utsagnskonstatene T og F er utsagn.
(Som logikere burde vi være mer forsiktige her, men vi skal ikke skille mellom en konstant og dens verdi i dette kurset.)
- Alle utsagnsvariable p_1, \dots, p_n er utsagn.
- Hvis p og q er utsagn, er $\neg p$, $(p \wedge q)$, $(p \vee q)$, $(p \rightarrow q)$ og $(p \leftrightarrow q)$ også utsagn.
En slik definisjon kaller vi en rekursiv eller induktiv definisjon.
Når vi gir en slik definisjon, begrenser vi bruken av ordet utsagn fra noe vagt, slik vi gjorde det innledningsvis, til noe matematisk presist.
Vi har en klar parallell i definisjonen av visse programmeringsspråk.
Vi skal komme tilbake til en mer systematisk drøfting av rekursive definisjoner senere i semesteret.

Oppbygging av utsagn

Den induktive oppbyggingen av utsagn forteller oss at vi har grunnutsagn og sammensatte utsagn, men også at noen utsagn er mer sammensatte enn andre. Når vi kommer til kapitlene om grafer og trær, vil vi se at et sammensatt utsagn kan betraktes som en trestruktur, hvor det gitte utsagnet ligger ved roten, og treet forgrener seg gjennom stadig mindre sammensatte delutsagn, helt til vi finner utsagnsvariablene ved bladene. Det første bindeordet vi kommer til når vi skal løse opp et utsagn i delutsagn kalles hovedkonnektivet eller, analogt med i boka, prinsipalkonnektivet. Dette illustreres på tavlen.

Mer om parenteser

Eksempel.

- $$(p \wedge q \rightarrow r) \rightarrow (p \rightarrow r) \vee (q \rightarrow r)$$
- Her mangler det noen parenteser, og for å kunne sette opp sannhetsverditabellen, må vi vite hvilke parenteser som mangler, eller, som er underforstått.
- Vi har tidligere sagt at \wedge og \vee skiller mer enn \neg .
- Vi skal også la \rightarrow og \leftrightarrow skille mer enn \wedge og \vee .
- Det betyr at utsagnet over egentlig skal være

$$(((p \wedge q) \rightarrow r) \rightarrow ((p \rightarrow r) \vee (q \rightarrow r)))$$

noe som ikke akkurat er lettere å lese.

Fortsettelse av eksemplet

Vi skriver ut trestrukturen til dette sammensatte utsagnet på tavlen og fortsetter med å skrive ut sannhetsverditabellen.

Vi oppdager at høyre søyle vil inneholde T i alle linjer.

Det betyr at utsagnet er sant uansett hvilke grunnutsagn vi setter inn for p , q og r .

Da må $(p \rightarrow r) \vee (q \rightarrow r)$ være en logisk konsekvens av $p \wedge q \rightarrow r$.

Fortsettelse av eksemplet

- La p stå for "Jeg betaler semesteravgift".
- La q stå for "Jeg får godkjent oblig'ene".
- La r stå for "Jeg kan gå opp til eksamen".
- Da er
 - Hvis jeg betaler semesteravgiften kan jeg gå opp til eksamen eller hvis jeg får godkjent oblig'ene kan jeg gå opp til eksamen.

en logisk konsekvens av

- Hvis jeg betaler semesteravgiften og får godkjent oblig'ene kan jeg gå opp til eksamen.

Er det noe galt her, og i så fall hva?:

Mer om parenteser

Hvordan skal vi forstå utsagn som

$$p \wedge q \wedge r$$

og

$$p \vee q \vee r?$$

I slike tilfeller vil vi få den samme høyresøylen i sannhetsverditabellen uansett hvordan vi setter parentesene, så vi kan like godt la det være.

Tautologier og kontradiksjoner

Definisjon.

La A være et sammensatt utsagn i utsagnsvariablene p_1, \dots, p_n .

A er en tautologi hvis A får verdien T for alle fordelinger av sannhetsverdier på p_1, \dots, p_n , det vil si hvis sannhetsverditabellen til A har bare T i høyre søyle.

Vi kunne brukt ordene selvpoppfyllende eller selvforklarende på norsk, men holder oss til det internasjonalt brukte tautologi.

Hvis sannhetsverdien til A derimot alltid blir F, kaller vi A en kontradiksjon eller en selvmotsigelse.

Tautologier og kontradiksjoner

Eksempel.

- $p \rightarrow (q \rightarrow p)$ er en tautologi.
- $(p \rightarrow q) \rightarrow ((q \rightarrow r) \rightarrow (p \rightarrow r))$ er en tautologi
- $p \wedge \neg p$ er en kontradiksjon
- $(p \leftrightarrow q) \wedge p \wedge \neg q$ er en kontradiksjon.
- $\neg p \rightarrow (p \rightarrow q)$ er en tautologi.

Logisk ekvivalens

Eksempel $(\neg p \wedge \neg q)$.

p	q	$\neg p$	$\neg q$	$\neg p \wedge \neg q$
T	T	F	F	F
T	F	F	T	F
F	T	T	F	F
F	F	T	T	T

Eksempel $(\neg(p \vee q))$.

p	q	$p \vee q$	$\neg(p \vee q)$
T	T	T	F
T	F	T	F
F	T	T	F
F	F	F	T

Vi ser at høyresøylene er identiske.

Logisk ekvivalens

Definisjon.

La A og B være to utsagnslogiske uttrykk.

Vi sier at A og B er logisk ekvivalente hvis A og B har samme sannhetsverdi uansett hvilke verdier vi gir til utsagnsvariablene.

Vi skriver

$$A \equiv B$$

når A og B er logisk ekvivalente.

Logisk ekvivalens

Eksempel.

- $\neg p \wedge \neg q \equiv \neg(p \vee q)$
- $\neg p \vee \neg q \equiv \neg(p \wedge q)$
- $\neg\neg p \equiv p$
- $p \rightarrow q \equiv \neg p \vee q$
- $p \rightarrow (q \rightarrow p) \equiv T$

Logisk ekvivalens

Tabell 4.12 på side 55 i læreboka lister opp en rekke regneregler for utsagnslogikk, kalt "laws of logic".

Poenget er at man kan regne på et uttrykk ved å bruke disse reglene på deluttrykk, for derved å prøve å forenkle det.

Det er et faktum vi ikke skal bevise (nå?) at vi kan regne oss frem til T fra enhver tautologi.

At disse lovene virkelig kan brukes som regneregler, bør bevises:

Logisk ekvivalens

Teorem.

La A være et sammensatt utsagn, og la B være et delutsagn av A.

La C være et annet utsagn slik at $B \equiv C$ og la D komme fra A ved at vi erstatter en eller flere forekomster av B med C.

Da er $A \equiv D$.

Bevis.

I sannhetsverditabellen for A har vi en søyle for B, og det er bare verdiene i denne søylen vi bruker videre.

Vi ville fått samme sluttresultat om vi hadde brukt en søyle for C i stedetfor.

Dette svarer til å sette opp sannhetsverditabellen for D.

Logisk konsekvens

- Logisk ekvivalens er et viktig begrep.
- Logisk konsekvens er et minst like viktig begrep:

Definisjon.

La A og B være sammensatte utsagn.

B er en logisk konsekvens av A dersom $A \rightarrow B$ er en tautologi.

Vi skriver ofte $A \Rightarrow B$ når B er en logisk konsekvens av A .

Merk at uttrykk som $A \equiv B$ og $A \Rightarrow B$ ligger på utsiden av den formelle utsagnslogikken.

Strategier

- Det finnes flere metoder eller strategier for å bestemme om et sammensatt utsagn er en tautologi eller ikke.
- Bruk av sannhetsverditabeller er en sikker, men tidkrevende metode.
- Vi skal ikke legge vekt på bruk av regnereglene for logikk her, men hvis man vil bruke den, kan man gjøre det målrettet, ved å eliminere \leftrightarrow og \rightarrow , flytte alle forekomster av \neg så langt inn som mulig og så bruke distributive lover og forkortningsregler.
- Hvis A er et sammensatt utsagn, kan vi "løse likningen" $A = F$ med hensyn på utsagnsvariablene.
Hvis likningen ikke har løsning, er A en tautologi.
- Hva som er mest hensiktsmessig avhenger av hvordan det sammensatte utsagnet ser ut.

Forelesning 7

Predikatlogikk

Dag Normann - 4. februar 2008

Oppsummering

- Vi har innført sannhetsverdiene T og F, begrepet utsagnsvariabel og de utsagnslogiske bindeordene \wedge , \vee , \neg , \rightarrow og \leftrightarrow .
- Vi har sett hvordan vi kan undersøke egenskapene til et sammensatt utsagn A ved å skrive ut sannhetsverditabellen til A.
- Et sammensatt utsagn A er en tautologi om A alltid får verdien T og A er en kontradiksjon om A alltid får verdien F.
- To sammensatte utsagn A og B er logisk ekvivalente om de alltid får den samme sannhetsverdien når vi gir sannhetsverdier til utsagnsvariablene.
- Dette er det samme som at $A \leftrightarrow B$ er en tautologi.
- Vi skriver $A \equiv B$ når A og B er logisk ekvivalente.

Oppsummering

- Vi diskuterte konvensjoner for å sette parenteser.
- Rekkevidden til \neg er det nærmeste deluttrykket som kan oppfattes som et utsagn. Vil vi at \neg skal rekke lenger enn til nærmeste utsagnsvariabel, må vi bruke parenteser.
- Rekkevidden til \wedge og \vee går til nærmeste symbol som ikke er \neg .
- Rekkevidden til \rightarrow og \leftrightarrow går frem til neste forekomst av \rightarrow eller \leftrightarrow , det vil si, forbi \neg , \wedge og \vee .
- Vi kan sette \wedge mellom mer enn to delutsagn uten å bruke parenteser, og vi kan sette \vee mellom mer enn to delutsagn uten å bruke parenteser, men bruker vi både \wedge og \vee må vi bruke parenteser for å bestemme hvilket som er hovedkonnektivet.
- Vi må alltid bruke parenteser hvis vi bruker flere \rightarrow eller \leftrightarrow etter hverandre, se eksempel.

Oppsummering

Eksempel ($p \rightarrow (q \rightarrow r)$).

- Dette utsagnet får verdien F når $p = T$ og $q \rightarrow r = F$.
- Det betyr igjen at det får verdien F nøyaktig når $p = T$, $q = T$ og $r = F$.

Eksempel $((p \rightarrow q) \rightarrow r)$.

- Dette utsagnet får verdien F når $p \rightarrow q = T$ og $r = F$.
- Dette utsagnet får altså verdien F når det første utsagnet får det, men også når $p = F$ og $r = F$ (uansett verdi på q .)

Utsagnene er altså ikke logisk ekvivalente.

Vi må bruke parentesene for å skille dem.

Oppsummering

- Vi refererte til side 55 i boka når det gjaldt regnereglene for logikk.
- Noen av de viktigste er:

– DeMorgans lover:

$$\neg(p \wedge q) \equiv \neg p \vee \neg q$$

$$\neg(p \vee q) \equiv \neg p \wedge \neg q$$

– Distributive lover:

$$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$$

$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$$

- Vi skal se på et eksempel på hvordan vi kan vise at et sammensatt utsagn er en tautologi ved å bruke disse regnereglene.
- Vi henviser til betegnelsene i tabellen på side 55.

Logisk ekvivalens

Eksempel $((p \vee q) \wedge (p \vee \neg q) \rightarrow p)$.

- $(p \vee q) \wedge (p \vee \neg q) \rightarrow p$
- $\neg((p \vee q) \wedge (p \vee \neg q)) \vee p$ [Eliminasjon av \rightarrow]
- $\neg(p \vee q) \vee \neg(p \vee \neg q) \vee p$ [Bruk av DeMorgan]
- $(\neg p \wedge \neg q) \vee (\neg p \wedge \neg \neg q) \vee p$ [To gangers bruk av DeMorgan]
- $(\neg p \wedge (\neg q \vee \neg \neg q)) \vee p$ [Distributiv lov]
- $(\neg p \wedge T) \vee p$ [Invers lov]
- $\neg p \vee p$ [Identitetsloven]
- T [Invers lov]

Logisk ekvivalens

- Vi antydnet også sist at det er mulig å vise at et sammensatt utsagn A er en tautologi ved å vise at likningen

$$A = F$$

er overbestemt.

- Vi skal gi ett eksempel på hvordan vi kan “løse slike likninger”.
- Metoden kan være nyttig når A har mange forekomster av \rightarrow .
- Programmeringsspråket *PROLOG* er basert på en systematisering av denne metoden, koblet med predikatlogikk.

Logisk ekvivalens

Eksempel $((p \rightarrow q) \rightarrow ((q \rightarrow r) \rightarrow (p \rightarrow r)))$.

- 1 $(p \rightarrow q) \rightarrow ((q \rightarrow r) \rightarrow (p \rightarrow r)) = F$
- 2 $p \rightarrow q = T$ (Fra 1.)
- 3 $(q \rightarrow r) \rightarrow (p \rightarrow r) = F$ (Fra 1.)
- 4 $q \rightarrow r = T$ (Fra 3.)
- 5 $p \rightarrow r = F$ (Fra 3.)
- 6 $p = T$ (Fra 5.)
- 7 $r = F$ (Fra 5.)
- 8 $q = F$ (Fra 4 og 7.)
- 9 $p = F$ (Fra 2 og 8)]
- 10 $p \neq p$ (Fra 6 og 9.)

Logisk ekvivalens

Oppgave.

- a) Vis at hvis A , B og C er sammensatte utsagn, så vil

$$(A \leftrightarrow B) \leftrightarrow C \equiv A \leftrightarrow (B \leftrightarrow C)$$

og at

$$(A \leftrightarrow B) \equiv (B \leftrightarrow A).$$

- b) Forklar hvorfor dette betyr at rekkefølge og parentessetting ikke betyr noe i et utsagnslogisk uttrykk som bare bruker bindeordet \leftrightarrow
- c) [Vanskelig] Hvordan kan vi lett avgjøre om et slikt uttrykk er en tautologi eller ikke?

Predikatlogikk

Utsagnslogikk er enkel i den forstand at gitt et utsagnslogisk uttrykk er det muligens tidkrevende, men i prinsippet enkelt, å avgjøre om vi står overfor en tautologi, en kontradiksjon eller noe annet.

Utfordringen i utsagnslogikk er å finne algoritmer som raskt kan løse denne typen problemstillinger for sammensatte utsagn (med mange utsagnsvariable) som forekommer i praktiske anvendelser.

Utsagnslogikk er også enkel i den forstand at den er uttrykksfattig, det er mange tilsynelatende logiske sluttninger som ikke kan presses inn i formatet til tautologier.

Vi skal starte med et eksempel:

Predikatlogikk

Eksempel.

Anta at vi vet følgende:

- All fluesopp er giftig.
- Det finnes sopp som ikke er giftig.

Da må vi ha lov til å konkludere med

- Det finnes sopp som ikke er fluesopp.

Eksempel.

- Vi vet følgende:

- Alle kvadrattall er ≥ 0 .
- Det finnes tall som ikke er ≥ 0

Da konkluderer vi med

- Det finnes tall som ikke er kvadrattall.

Dette er det samme argumentet i to forkledninger.

Predikatlogikk

Da vi innledet utsagnslogikken definerte vi et predikat som en ytring med variable, som ville bli sann eller usann hver gang vi gir variablene verdier.

I det første eksemplet kan vi betrakte *soppen* som en variabel som kan ta en hvilken som helst sopp som verdi.

Da blir *soppen er giftig* og *soppen er en fluesopp* predikater.

I det andre eksemplet er *tall* en variabel som kan ta alle hele tall som verdi. Da er *tallet er et kvadrattall* og *tallet er ≥ 0* predikatene.

Det gjennstår å betrakte uttrykk som *alle sopper* og *det finnes tall* som en del av en utvidet logisk struktur.

Predikatlogikk

Eksempel.

La $f : [a, b] \rightarrow \mathbb{R}$ være en funksjon.

Hvordan skal vi uttrykke

f har et minimumspunkt?

Løsning

Det finnes $x \in [a, b]$ slik at for alle $y \in [a, b]$ vil $f(x) \leq f(y)$.

Trangen til å finne egne symboler for det finnes og for alle virker snart påtrengende.

Predikatlogikk

Vi ser på et eksempel til:

Det finnes ikke noe største primtall

Vi prøver med litt utsagnslogikk:

$\neg(\text{Det finnes et største primtall}),$

det vil si at det er ikke slik at det finnes et primtall som er større eller lik alle primtallene.

Vi trenger litt formelt språk for å få orden på dette!!!

Kvantorer

Definisjon.

- Hvis P er et predikat og x er en variabel, vil

$\exists xP$

uttrykke at det finnes en verdi av x slik at P holder.

$\forall xP$

uttrykker at P holder for alle verdier x kan ha.

Vi kaller \exists og \forall for kvantorer, og vi regner dem som en del av det formelle logiske vokabularet.

Kvantorer

Eksempel.

a)

$$\exists x(x \in [a, b] \wedge \forall y(y \in [a, b] \rightarrow f(x) \leq f(y)))$$

uttrykker at det finnes et minimumspunkt for f på $[a, b]$.

b)

$$\neg \exists x(x \text{ primtall} \wedge \forall y(y \text{ primtall} \rightarrow y \leq x))$$

uttrykker at det ikke finnes et største primtall.

Kvantorer

Det kan være lurt å øve seg på å skrive uttalelser i dagligtale om til utsagn med kvantorer, men for det meste vil vi bruke kvantorer når vi trenger matematisk presisjon i matematikk eller informatikk.

Vi skal se på noen eksempler på hvordan man oversetter fra dagligtale til formalspråk og omvendt.

Flere eksempler finnes i læreboka.

Kvantorer

Eksempel.

- *Alle hunder har lopper, men ikke alle hunder har lus.*

$$\forall x (\text{hund } x \rightarrow \exists y(\text{loppe } y \wedge x \text{ har } y)) \wedge \neg \forall x (\text{hund } x \rightarrow \exists y(\text{lus } y \wedge x \text{ har } y))$$

- *Alle har et søskenbarn på Gjøvik.*

$$\forall x \exists y (y \text{ bor på Gjøvik og } y \text{ er søskenbarn til } x)$$

- *Ingen er bedre enn Tor til å fiske laks*

$$\neg \exists x (x \text{ er bedre enn Tor til å fiske laks})$$

Kvantorer

Eksempel.

- $\forall x \forall y (\exists z (\text{far}(z, x) \wedge \text{far}(z, y)) \rightarrow \text{brødre}(x, y))$

Hvis to personer har en felles far, er de brødre.

Dette er selvfølgelig ikke sant.

- $\forall x \exists y (x \text{ har slått } y \wedge y \text{ har slått } x)$

Dette dreier seg om fotball-lag.

For alle lag finnes det et annet lag slik at de har slått hverandre.

- $\neg \forall x \exists y (y \text{ er bestevennen til } x)$

Ikke alle har en bestevenn.

Kvantorer

Eksempel.

a) $\exists x \forall y (x \leq y)$

b) $\forall y \exists x (x \leq y)$

- Rekkefølgen vi skriver kvantorene i betyr mye for hva utsagnet sier:
 - a) sier at det finnes et minste objekt.
 - b) sier at det alltid finnes et objekt som er mindre eller lik.
- Hvis x varierer over de hele tallene er a) feil, mens b) holder.
- Hvis x varierer over de naturlige tallene, holder a).
- b) holder også, fordi gitt en verdi for y kan vi bruke samme verdi for x .
- Før vi kan bestemme om et utsagn med kvantorer er sant eller usant, må vi vite hvilke mulige verdier variablene kan ta.
- I en programmeringssammenheng vil vi alltid deklare datatypen til en variabel, og da kan variabelen ta alle verdier i denne datatypen.

Kvantorer

Når er et utsagn med kvantorer logisk holdbart?

La oss betrakte følgende eksempel:

Eksempel.

$$\forall x \forall y \forall z ((x < y \wedge y < z) \rightarrow x < z).$$

- Selv om vi ikke har bestemt oss for hvilke verdier x , y og z kan ta, uttrykker dette en sammenheng som vi mener er underforstått når vi bruker symbolet $<$.
- La x , y og z variere over lagene i en avdeling i en fotball- eller håndball-liga.
- Hvis L_1 og L_2 er to lag, sier vi at $L_1 < L_2$ hvis L_1 ble dårligere enn L_2 i de innbyrdes oppgjørene.
- Det er ofte at vi kan finne tre lag som "slår hverandre".
- I den situasjonen er ikke utsagnet over sant.

Kvantorer

Definisjon.

- Et predikat er en ytring

$$P(x_1, \dots, x_n)$$

hvor det kan forekomme variable.

- Hvis P er et predikat og x er en variabel, er $\exists xP$ og $\forall xP$ nye predikater hvor variabelen x er bundet.
- Variable som ikke er bundet kalles fri
- Hvis vi setter inn (lovlige) verdier for de frie variablene i et predikat får vi et utsagn.
- For å bestemme om et utsagn er sant eller usant må vi bestemme variasjonsområdene til alle variablene samt hva andre symboler skal stå for.

Kvantorer

Definisjon (fortsatt).

- En setning er et predikat uten frie variable. Dette kalles også ofte for et lukket utsagn.
- En setning er logisk gyldig dersom den er sann uansett hvilke variasjonsområder vi velger og uansett hva vi lar symbolene bety.

Denne definisjonen er ikke matematisk sett helt presis, men den holder for vårt formål

Kvantorer

Eksempel.

- $x < y \rightarrow \neg(y < x)$ er et predikat med to frie variable x og y .
- $\exists x(x < y \rightarrow \neg(y < x))$ er et predikat med en fri variabel y og en bunden variabel x .
- $\forall y \exists x(x < y \rightarrow \neg(y < x))$ er en setning, fordi begge variablene er bundne.
- For å bestemme om denne setningen er sann eller usann, må vi bestemme oss for hvilke verdier x og y kan ta, og for hva vi mener med $x < y$.
- Hvis vi lar x og y variere over \mathbb{Z} og $<$ være vanlig ordning, viser vi at setningen er sann på vanlig matematisk måte:

Kvantorer

Eksempel ($\forall y \exists x (x < y \rightarrow \neg(y < x))$).

- Beviset kan formuleres slik;

La y få en vilkårlig verdi a

La x også få verdien a .

Siden $a < a$ er usant, må $\neg(a < a)$ være sant, og sannhetsverdien til

$$x < y \rightarrow \neg(y < x)$$

blir T når vi setter inn a for både x og y .

Merk at a var vilkårlig da vi satte a inn for x , men valgt med omhu da vi satte a inn for y .

- Dette gir oss ingen grunn til å mene at setningen er logisk gyldig.

Kvantorer

- Ved hjelp av læreboka listet vi opp en rekke regneregler for utsagnslogikk.
- Det finnes tilsvarende regler for regning med uttrykk med kvantorer.
- En alternativ måte er å isolere noen utsagn i predikatlogikk som aksiomer og fastsette noen regler for hvordan man kan bevise andre utsagn fra disse aksiomene.
- Dette er noe som taes opp på et senere trinn i emner både ved Institutt for Informatikk og ved Matematisk Institutt.
- Vi skal se på et par regneregler som vil være utledbare i en slik logikk, men hvor vi kan overbevise oss om gyldigheten her og nå.
- Vi definerte \equiv som en relasjon mellom utsagnslogiske utsagn, men vil utvide bruken til utsagn med kvantorer, når utsagnene åpenbart er sanne under nøyaktig de samme omstendighetene.

Kvantorer

Eksempel (DeMorgans lover for kvantorer).

For alle utsagn A og variable x vil

1 $\neg \forall x A \equiv \exists x \neg A$

2 $\neg \exists x A \equiv \forall x \neg A$

Noen ganger kan det være lettere å argumentere for en abstrakt påstand ved å gi et dekkende eksempel.

Vi kan argumentere for 1 ved følgende eksempel som dekker alle andre eksempler:

Vi mener det samme når vi sier

- Det er feil at alle russere er katolikker.

- Det finnes en russer som ikke er katolikk.

Vi kan argumentere for 2 ved følgende eksempel:

Vi mener det samme når vi sier

- Det finnes ingen ærlig politiker.
- For alle politikere gjelder det at de ikke er ærlige.

Kvantorer

Eksempel (Sammentrekning av kvantorer).

For alle utsagn A og B gjelder

$$1 \quad \exists xA \vee \exists xB \equiv \exists x(A \vee B)$$

$$2 \quad \forall xA \wedge \forall xB \equiv \forall x(A \wedge B)$$

- Om vi sier

Det finnes en elev i klassen som spiller tennis eller det finnes en som spiller badminton
mener vi det samme som om vi sier

Det finnes en elev i klassen som spiller tennis eller badminton.

Kvantorer

- Om vi sier

Alle arbeiderne fikk høyere lønn og alle arbeiderne fikk kortere arbeidstid
mener vi det samme som om vi sier

Alle arbeiderne fikk høyere lønn og kortere arbeidstid

Igjen er disse eksemplene dekkende for den generelle situasjonen.

Kvantorer

- Det er ertVIKTIG at man ikke trekker $\exists x$ over en \wedge eller en $\forall x$ over en \vee .

Eksempel (To moteksempler).

- Utsagnet

Noen Nordmenn er mangemillionærer og noen Nordmenn lever under fattigdomsgrensen
er på formen

$$\exists xM(x) \wedge \exists xUF(x).$$

Utsagnet

$$\exists x(M(x) \wedge UF(x))$$

uttrykker at noen Nordmenn både er millionærer og samtidig lever under fattigdomsgrensen.

Kvantorer

Eksempel (To moteksempler, fortsatt).

Den første påstanden er nok sann, mens den andre er heller tvilsom.

Det betyr at de to utsagnene ikke er logisk ekvivalente.

Kvantorer

Eksempel (To moteksempler, fortsatt).

- Utsagnet

Alle barna får tilbud om å stå slalåm eller å gå langrenn

er på formen

$$\forall x(S(x) \vee L(x)).$$

Utsagnet

$$\forall xS(x) \vee \forall xL(x)$$

sier at det er det samme tilbudet til alle barna, mens det første utsagnet gir muligheten for at det er et valg.

Utsagnene er derfor ikke logisk ekvivalente.

Forelesning 8

Predikatlogikk, bevisføring

Dag Normann - 6. februar 2008

Kvantorer

- Mandag 04.02.2008 introduserte vi predikatlogikk
- Vi innførte
 - eksistenskvantoren \exists
 - allkvantoren \forall
- Vi så på en del eksempler på oversettelse mellom dagligtale og uttrykk med kvantorer.
- Vi viste noen logiske ekvivalenser:
 - deMorgans lover: $\neg\exists xA \equiv \forall x\neg A$ og $\neg\forall xA \equiv \exists x\neg A$.
 - Sammentrekning: $\exists xA \vee \exists xB \equiv \exists x(A \vee B)$ og $\forall xA \wedge \forall xB \equiv \forall x(A \wedge B)$.

Kvantorer

Eksempel.

- Den tekniske definisjonen av at en funksjon f er kontinuert i et punkt x er:

$$\forall\epsilon\exists\delta\forall y(\epsilon > 0 \rightarrow \delta > 0 \wedge (|x - y| < \delta \rightarrow |f(x) - f(y)| < \epsilon))$$

Hvis vi skal uttrykke at f ikke er kontinuert i x må vi negere denne setningen.

I første omgang bruker vi deMorgans lover for kvantorene, og får

$$\exists\epsilon\forall\delta\exists y\neg(\epsilon > 0 \rightarrow \delta > 0 \wedge (|x - y| < \delta \rightarrow |f(x) - f(y)| < \epsilon))$$

Kvantorer

Eksempel (Fortsatt).

Ved deretter å bruke reglene for utsagnslogikk kan vi skrive om

$$\neg(\epsilon > 0 \rightarrow \delta > 0 \wedge (|x - y| < \delta \rightarrow |f(x) - f(y)| < \epsilon))$$

til

$$\epsilon > 0 \wedge (\delta \leq 0 \vee (|x - y| < \delta \wedge |f(x) - f(y)| \geq \epsilon))$$

Vi har tillatt oss å skrive \geq i stedet for $\neg <$.

Hele uttrykket blir da

$$\exists \epsilon \forall \delta \exists y (\epsilon > 0 \wedge (\delta \leq 0 \vee (|x - y| < \delta \wedge |f(x) - f(y)| \geq \epsilon)))$$

Kvantorer

Eksempel (Fortsatt).

Det er usikkert om noen får lyst til å studere analyse etter dette.

Det illustrerer imidlertid at det krever god kontroll over bruk av kvantorer og konnektiver å kunne finne ut av hva det betyr at en viktig matematisk definisjon ikke holder i en gitt situasjon.

Det illustrerer også at det kan gi bedre leselighet om vi "flytter" noe av det som uttrykkes gjennom utsagnslogikk til en begrensning av virkeområdet til kvantoren:

$$\forall \epsilon > 0 \exists \delta > 0 \forall y (|x - y| < \delta \rightarrow |f(x) - f(y)| < \epsilon)$$

hvor negasjonen blir

$$\exists \epsilon > 0 \forall \delta > 0 \exists y (|x - y| < \delta \wedge |f(x) - f(y)| \geq \epsilon).$$

Relevans?

- Et naturlig spørsmål nå vil være om predikatlogikk har noen relevans for informatikk.
- Det er ikke naturlig å bruke kvantorer i test-uttrykk i pseudokoder, kontrollstrukturer eller i programmeringsspråk bygget over pseudokodefilosofien.
- Grunnen er at det generelt ikke finnes noen algoritme for å bestemme om en setning er sann eller usann.
- Hvis kvantorene skal variere over data lagret i en base, trenger ikke sannhetsverdiene til utsagn med kvantorer å være stabile.
- Vi skal se på to eksempler som antyder hvordan bruk av predikater og til dels kvantorer kan være nyttige i en informatikk-sammenheng.

Relevans?

Eksempel (Kvalitetssikring av databaser).

- Anta at vi skal bygge opp en base for registrering av slektskapsforhold.
- Vi vil registrere noen grunnleggende slektskapsforhold.
- For å sikre oss mot at vi lagrer data på feil måte skal vi sette opp visse aksiomer som dataene våre skal respektere.

- Hvis vi kan utlede en kontradiksjon fra de lagrede slektskapsforholdene og aksiomene, har vi foretatt en feil-lagring.

Relevans?

Eksempel (Fortsatt).

- Noen aksiomer kan være
 - $\text{Far}(x, y) \wedge \text{Far}(x, z) \rightarrow \text{Søsken}(y, z)$
 - $\text{Mor}(x, y) \wedge \text{Mor}(x, z) \rightarrow \text{Søsken}(y, z)$
 - $\text{Søsken}(x, y) \wedge \text{Far}(x, z) \wedge \text{Mor}(y, z) \rightarrow \text{F}$
- Dette sikrer at vi ikke lagrer søsken som foreldre til det samme barnet.
- Riktignok medfører disse aksiomene at alle er sin egen søsken, men siden ingen av oss vil kunne bli både mor og far til det samme barnet, er kvalitetssikringen ivaretatt uansett.

Relevans?

- Hvis en datamaskin skal gi oss en feilmelding ut fra at de dataene vi har lastet inn leder til en kontradiksjon, må den være programmert til å gjøre det.
- En mulighet er å bruke et spesialkonstruert programmeringsspråk PROLOG til dette formålet.
- Et PROLOG-program vil være en liste av kvantorfrie predikater av en bestemt form, og når programmet kjøres innebærer det å vise at predikatene samlet sett er kontradiktoriske.

Relevans?

- PROLOG har sin egen syntaks, men oversatt til vårt språk kan en PROLOG-instruks være på en av tre former:
 1. $A_1 \wedge \dots \wedge A_n \rightarrow B$
 2. $A_1 \wedge \dots \wedge A_n \rightarrow \text{F}$
 3. B
- Svært mange relevante sammenhenger mellom data i en base kan formuleres som en PROLOG-instruks.
- Her vil A_i og B være positive grunnpredikater uten bruk av kvantorer eller bindeord, da heller ikke negasjon.
- Eksempler kan være $x < y$, $\text{Far}(x, y)$, $\text{Forbudt}(\text{Promillekjøring})$ og "Per bedriver promillekjøring".
- Vi skal gi et veldig enkelt eksempel på hvordan PROLOG kan brukes til å søke etter data i en base:

Relevans?

Eksempel.

- Vi ser litt nærmere på slektskapsbasen vi så på i sted.
- Vi har lagret informasjon om hvem som er mor eller far til hvem. Annen informasjon må utledes.
- På samme måte som vi innfører *Søsken* som en utledet egenskap, kan vi innføre *Farfar* ved
$$\text{Far}(x, y) \wedge \text{Far}(y, z) \rightarrow \text{Farfar}(x, z),$$
- Tor oppsøker denne basen og lur på om han har noen farfar.

Relevans?

Eksempel (Fortsatt).

- I basen er det lagret $\text{Far}(\text{Per}, \text{Tor})$ og $\text{Far}(\text{Knut}, \text{Per})$.
- Programmereren som styrer basen legger til aksiomet

$$\text{Farfar}(x, \text{Tor}) \rightarrow F$$

- Dette sier at x ikke er farfar til Tor.
- Hvis det leder til en motsigelse, vet Tor at han har en farfar registrert i basen.

Relevans?

Eksempel (Fortsatt).

- PROLOG vil nå målrettet prøve å utlede F fra det nye aksiomet og den lagrede informasjonen.
- Gangen vil være omtrent som følger:
 - 1 Fra $\text{Farfar}(x, \text{Tor}) \rightarrow F$ og $\text{Far}(x, y) \wedge \text{Far}(y, z) \rightarrow \text{Farfar}(x, z)$ kan vi slutte $\text{Far}(x, y) \wedge \text{Far}(y, \text{Tor}) \rightarrow F$ ved at vi setter inn Tor for z .
 - 2 Fra $\text{Far}(x, y) \wedge \text{Far}(y, \text{Tor}) \rightarrow F$ og $\text{Far}(\text{Per}, \text{Tor})$ kan vi slutte $\text{Far}(x, \text{Per}) \rightarrow F$ ved at vi setter inn Per for y .
 - 3 Fra $\text{Far}(\text{Knut}, \text{Per})$ og $\text{Far}(x, \text{Per}) \rightarrow F$ kan vi slutte F ved at vi setter inn Knut for x .
- PROLOG vil ikke bare bevise at Tor har en farfar, men den virker slik at den finner frem til en farfar blant dataene.
- For de som bare leser utskriftene: Endel ble forklart muntlig på forelesningen.

Logikk, oppsummering

Læringsmålene for kapitlet om logikk er:

1. Definisjonene av utsagn og predikat, og å kunne bestemme hvilke ytringer som er utsagn, hvilke som er predikat og hvilke som faller utenfor rammene våre.
2. De logiske bindeordene \neg , \wedge , \vee , \rightarrow og \leftrightarrow og hvordan de defineres via sannhetsverditabeller.
3. Sette opp sannhetsverditabellen til et sammensatt uttrykk, og bruke denne til å bestemme om et utsagn er en tautologi, en kontradiksjon eller ingen av delene.
4. Kjenne til logikkens lover og i noen utstrekning kunne bruke dem.
5. Spesielt sentralt står deMorgans lover og de distributive lovene.
6. Kjenne definisjonene av kvantorene \forall og \exists og kjenne deMorgans lover for kvantorer.
7. Kunne uttrykke en sammenheng ved bruk av kvantorer og kunne "forstå" et uttrykk som inneholder kvantorer.

Bevisteknikker

- Den siste delen av kapittel 4 hører egentlig ikke inn under en overskrift "Logikk".
- Vi skal se på måter å strukturere et matematisk bevis på.
- Dette er et tema alle studenter i matematikk eller et annet teoretisk fag etterhvert vil kjenne seg igjen i.
- Vi skal se på direkte bevis, bevis ved tilfeller og kontrapositive bevis.
- Senere skal vi føye induksjonsbevis til vår meny av bevisteknikker.
- Vi skal eksemplifisere de forskjellige bevisformene.

Bevisteknikker

Eksempel.

- Vi skal vise at differensen mellom to kvadrattall som kommer etter hverandre i tallrekken er et oddetall.
- Vi kan formulere dette mer matematisk som en setning:
 - For alle tall n er $(n + 1)^2 - n^2$ et oddetall.

Bevis.

Ved 1. kvadratsetning er $(n + 1)^2 = n^2 + 2n + 1$, så $(n + 1)^2 - n^2 = n^2 + 2n + 1 - n^2 = 2n + 1$. Siden $2n + 1$ alltid er et oddetall er setningen vist.

Bevisteknikker

- I dette eksemplet formulerte vi først det vi skulle vise i en matematisk språkdrakt, deretter regnet vi litt på den differensen vi skulle bevise var et oddetall, og endte opp med at det var akkurat et oddetall det var.

- Hvis vi analyserer beviset litt nærmere ser vi at alle oddetallene kan fremkomme som en slik differens, for å få $2n + 1$ som verdi, kan vi velge kvadrattallene $(n + 1)^2$ og n^2 .
- Når vi først har funnet et bevis, kan vi undersøke om samme metode kan gi oss mere innsikt. Det vil ofte være tilfelle, men kan kreve ekstra innsats. La oss se om vi kan bruke samme resonnement til å si noe om differensen mellom kubikktall.

Bevisteknikker

Teorem.

- For alle naturlige tall n er $(n + 1)^3 - n^3$ et oddetall.

Bevis.

Vi har at $(n + 1)^3 = n^3 + 3n^2 + 3n + 1$, så

$$(n + 1)^3 - n^3 = 3n^2 + 3n + 1.$$

Hvis n er et partall, er $3n^2 + 3n$ også et partall, så $(n + 1)^3 - n^3$ er et oddetall.

Hvis n er et oddetall, er både $3n^2$ og $3n$ oddetall, så $3n^2 + 3n$ er fortsatt et partall, og også i dette tilfellet er $(n + 1)^3 - n^3$ et oddetall.

Dermed er påstanden bevist.

Bevisteknikker

Som vi ser, brukte vi akkurat samme resonnement i starten av disse to bevisene.

I det andre beviset måtte vi imidlertid etterhvert dele argumentet opp i to tilfeller, ett for at n er et oddetall og ett for at n er et partall. Siden dette dekker alle mulighetene, er beviset fullstendig.

La oss gi et tredje eksempel på et bevis hvor vi må dele argumentet opp i tilfeller:

Bevisteknikker

Eksempel.

La oss bevise følgende påstand:

Hvis n er et helt tall, kan $n^2 - n$ deles på 6 eller $n^2 + n$ kan deles på 6.

Bevisteknikker

Bevis.

Vi har at $n^2 - n = (n - 1)n$ og at $n^2 + n = n(n + 1)$.

Nøyaktig ett av tallene $n - 1$, n eller $n + 1$ kan deles på 3.

Hvis $n - 1$ kan deles på 3 er ett av tallene $(n - 1)$ eller n et partall, og da er $(n - 1)n$ delelig med 6.

Hvis $n + 1$ er delelig med 3 er ett av tallene n eller $n + 1$ partall, så $n(n + 1)$ er delelig med 6.

Hvis n er delelig med 3 ser vi ved samme argument at både $(n - 1)n$ og $n(n + 1)$ er delelige med 6.

Tilsammen beviser dette påstanden.

Bevisteknikker

- Dette eksemplet viser hvordan man enkelte ganger må dele et argument opp i tilfeller.
- Det viser imidlertid også at man av og til må få anta at leseren henger med i noen av svingene, uten at alle detaljene som ligger til grunn for beviset blir tatt med.
- Vi har for eksempel tatt det for gitt at leseren er med på at 6 er faktor i $(n - 1)n$ når $n - 1$ kan deles på 3 og $n - 1$ eller n er et partall.
- Vi har heller ikke minnet om det er fordi $n^2 - n = (n - 1)n$ at vi har vist påstanden når vi i realiteten viser at $(n - 1)n$ eller $n(n + 1)$ kan deles på 6.
- Hvor mange detaljer man tar med er en vurderingsak, og vil være avhengig av målgruppen.

Bevisteknikker

- De bevisene vi har sett på til nå kalles direkte bevis
- Dette er for å skille dem fra de såkalte kontrapositive bevisene.
- Hvis vi går tilbake til utsagnslogikken, ser vi at utsagnene

$$p \rightarrow q$$

og

$$\neg q \rightarrow \neg p$$

er logisk ekvivalente.

- Det betyr at hvis vi ønsker å vise en påstand på formen $A \rightarrow B$, kan vi like gjerne anta $\neg B$ og bevise $\neg A$.
- Et bevis på den formen kalles kontrapositivt.

Bevisteknikker

- Før vi gir et eksempel, skal vi se på et generelt spesialtilfelle:
- Det å bevise en påstand A er det samme som å vise $T \rightarrow A$.

- Den kontrapositive varianten vil være å vise $\neg A \rightarrow \neg T$, det vil si $\neg A \rightarrow F$.
- F er selvmotsigelsen i sin reneste form, så en måte å bevise A på vil være å anta $\neg A$, og så utlede en selvmotsigelse.

Bevisteknikker

- Det klassiske eksemplet på et kontrapositivt bevis er Pythagoreernes argument for at $\sqrt{2}$ ikke er et rasjonalt tall.
- Dette er gitt som en oppgave i boka, og vi skal la det være med det.
- Vi skal gi to eksempler på kontrapositive bevis.
- Det ene er bare for å illustrere metoden, mens det andre viser et meget viktig resultat som berører forståelsen av informatikkens begrensninger.

Bevisteknikker

Teorem.

Alle naturlige tall > 1 er primtall eller kan faktoriseres i primtall.

Bevis.

Anta at $n > 1$ hverken er et primtall eller kan faktoriseres i primtall.

Siden n ikke er et primtall, kan n skrives som et produkt $n = ab$ hvor $1 < a < n$ og $1 < b < n$.

Hvis både a og b enten er primtall eller kan faktoriseres i primtall, vil n kunne faktoriseres i primtall.

Bevisteknikker

Bevis (Fortsatt).

Derfor finnes det $n_1 < n$ slik at $1 < n_1$ og slik at n_1 hverken er et primtall eller kan faktoriseres i primtall.

Da kan vi fortsette dette resonementet n ganger, og får $1 < n_n < n_{n-1} < \dots < n_1 < n$ slik at ingen av disse tallene er primtall eller kan faktoriseres i primtall.

Dette er umulig, siden det ikke finnes så mange tall mindre enn n .

Derfor må forutsetningen være feil.

Når vi har lært om induksjonsbevis, vil vi kunne bevise dette på en måte som virker mer overbevisende på matematikere.

Bevisteknikker

Vi skal vise at under svært generelle forutsetninger er det ikke mulig å løse noen helt grunnleggende problemer om egenskaper ved programmer.

Vi skal la \mathcal{P} være et programmeringsspråk som har følgende egenskaper:

- Det er mulig å skrive et program for en prosedyre som i teorien aldri stopper (mens $x \geq 0$, sett $x = x + 1$).
- Vi kan la et program P_2 etterfølge et program P_1 ved en konstruksjon som

$$P_1; P_2.$$

- Vi kan skille mellom tilfeller som i **If ... then ... else**.
- En hvilken som helst tekstfil, eksempelvis et program, kan tjene som input.
- Det finnes et program for kopiering av en fil, det vil si, som til en inputtekst t gir output tt , dvs t og en kopi av t .

Bevisteknikker

Teorem.

La \mathcal{P} være et programmeringsspråk med egenskapene på forrige side.

Da finnes det ikke noe program Q for å avgjøre om et annet program P med input t vil stoppe eller fortsette i det uendelige.

Bevis.

Med et program vil vi her mene tekstfilen som utgjør programmet.

Hvis P er et program og t er et mulig input, skriver vi $P(t)$ for tilsvarende output.

Anta at teoremet er feil, og at det finnes et program Q slik at om P er et annet program og t er et mulig input, så vil

- $Q(Pt) = 1$ om $P(t)$ har en verdi, dvs P med input t stopper.
- $Q(Pt) = 0$ om $P(t)$ i teorien aldri stopper.

Bevisteknikker

Bevis (Fortsatt).

La C være et program slik at $C(t) = tt$ for alle t og la U være et program som aldri stopper uansett input.

La R være et program som svarer til

If $Q(C(t)) = 1$ **then** U **else** output 1.

Husk at R også er tekstfilen til R .

Anta at $R(R)$ stopper.

Da vil $Q(C(R)) = Q(RR) = 1$ fordi $R(R)$ stopper.

Men etter den testen, fortsetter R med U , det vil si at $R(R)$ ikke stopper.

Bevisteknikker

Bevis (Fortsatt).

Men da er $Q(C(R)) = 0$, så R gir output 1, hvilket betyr at $R(R)$ stopper likevel.

Når vi konstruerer R på denne måten, vil $R(R)$ stoppe hvis og bare hvis beregningen aldri stopper.

Dette er en motsigelse, og konklusjonen må være at det ikke finnes noe program Q som følger spesifikasjonene.

Bevisteknikker

I informatikk kan det ofte være lurt å bruke konstruktive bevis.

Det betyr at man ikke ukritisk gjør bruk av tautologier som $p \vee \neg p$, men at man skal ha kontroll på hvilken av de to delene som holder.

Kontrapositive bevis har heller ingen plass i konstruktiv matematikk.

Fordelen med konstruktive bevis er at man kan trekke algoritmer og annen form for informasjon ut av bevisene.

Det klassiske beviset for at det alltid finnes et større primtall er konstruktivt.

Vi skal se et eksempel på et bevis som ikke er konstruktivt.

Bevisteknikker

Teorem.

Det finnes to irrasjonale tall a og b slik at $a^b \in \mathbb{Q}$.

Bevis.

Vi deler beviset opp i to tilfeller.

Tilfelle 1: $\sqrt{2}^{\sqrt{2}} \in \mathbb{Q}$.

Da lar vi $a = b = \sqrt{2}$ og $a^b \in \mathbb{Q}$.

Tilfelle 2: $\sqrt{2}^{\sqrt{2}} \notin \mathbb{Q}$.

Da lar vi $b = \sqrt{2}$ og $a = \sqrt{2}^{\sqrt{2}}$.

Da får vi

$$a^b = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{(\sqrt{2} \cdot \sqrt{2})} = \sqrt{2}^2 = 2 \in \mathbb{Q}.$$

Bevisteknikker

- Dette eksemplet blir ofte brukt til å illustrere forskjellen på konstruktive bevis og bevis basert på klassisk logikk.
- I klassisk logikk kan vi gjøre uhemmet bruk av antagelsen at enten gjelder en påstand P eller så gjelder negasjonen $\neg P$.
- I konstruktiv matematikk kreves det at vi i tillegg har noe informasjon om hvilken av de to som gjelder, eller i det minste en metode for å avgjøre hvilken av de to som gjelder i en gitt situasjon.
- Ut fra det beviset vi har sett på kan vi ikke si noe sikkert om hvilket par a, b av irrasjonale tall det er som er slik at $a^b \in \mathbb{Q}$, bare at det finnes et slikt par.

Forelesning 9

Mengdelære

Dag Normann - 11. februar 2008

OVER TIL KAPITTEL 5

Mengder

- De fleste som tar MAT1030 har vært borti mengder i en eller annen form tidligere.
- I statistikk og sannsynlighetsteori på VGS behandler man utfallsrom og studerer matematiske sannsynligheter eller sannsynlighetsfordelinger basert på eksperimenter på slike utfallsrom.
- Man ser på delmengder av slike utfallsrom og eksempelvis Bayes setning omhandler både det vi vil kalle komplement og det vi vil kalle snitt (og med de betegnelsene).
- Mengdebegrepet brukes også i de innledende emnene på universitetet, eksempelvis i form av løsningsmengder for ulikheter.

Mengder

- En mengde er en samling objekter hvor det er entydig bestemt hvilke objekter som er med i mengden eller ikke.
- Bruken av mengder gjennomsyrrer matematikk og andre teoretiske fag.
- Vi skal lære å bruke mengder slik at vi kan uttrykke oss presist om konstruksjoner og begreper av interesse i informatikk i en vid forstand.
- Vi bruker klammeparenteser $\{$ og $\}$ for å beskrive mengder. Vi skal illustrere bruken ved eksempler.

Mengder

Eksempel.

- $\{0, 1\}$ er mengden av digitale verdier en bit kan ha.
- $\{T, F\}$ er mengden av sannhetsverdier.
- $\{2, 3, 5, 7, 11, 13, 17, 19, 23, 29\}$ er mengden av de 10 minste primtallene.
- $\mathbb{N} = \{1, 2, 3, \dots\}$ er mengden av naturlige tall

Mengder

- I de tre første eksemplene har vi listet opp mengdens elementer. Mengdene er endelige, og i disse tilfellene så små at vi kan beskrive mengden ved hjelp av en liste med

klammeparenteser rundt.

- Vi sier at vi har gitt mengden på listeform.
- I en viss forstand har vi beskrevet \mathbb{N} ved hjelp av en liste. I dette tilfellet er listen uendelig lang, og vi har brukt prikkene \dots for å antyde at oppstillingen fortsetter.
- Denne mengden er også gitt på listeform.
- Bruker vi listeform med prikker eller tilsvarende, må vi være sikre på at leseren vil oppfatte prikkene på samme måte som forfatteren.
- Hvordan ser neste element i mengden $\{1, 5, 15, 34, 65, \dots\}$ ut?
- Hjelper det å få vite at det er 111?

Mengder

Vi trenger et eget symbol for å uttrykke at et objekt er et element i en mengde:

Definisjon.

- a) Vi skriver $a \in A$ for å uttrykke at a er et element i mengden A .
- b) Vi skriver $a \notin A$ for å uttrykke at a *ikke* er et element i A .

Vi kunne ha skrevet $\neg(a \in A)$ i stedetfor.

Det er ikke uvanlig å bruke \neq som nektingsymbol, som i $3 \neq 2$ og $4 \neq 3$.

Mengder

Hvis vi skal beskrive noen mer kompliserte mengder, må vi bruke et litt annet format enn listeform.

Dette illustreres best ved noen eksempler:

Eksempel.

- $\{n \in \mathbb{N} : n \text{ kan deles på } 5\}$
- $\{n \in \mathbb{N} : n \text{ er et primtall}\}$

Eksempel.

- $\{n \in \mathbb{N} : \exists k \in \mathbb{N}(n = 5k)\}$
- $\{n \in \mathbb{N} : n \geq 2 \wedge \forall m \in \mathbb{N} \forall k \in \mathbb{N}(n = km \rightarrow k = 1 \vee m = 1)\}$

Mengder

- Noen mengder vil vi referere til så ofte at vi bruker egne symboler for dem.
- Vi har allerede sett at vi bruker \mathbb{N} for mengden av naturlige tall.
- Vi lar $\mathbb{J} = \{\dots, -3, -2, -1, 0, 1, 2, 3, 4, \dots\}$ være mengden av hele tall.
Her følger vi boka, men internasjonalt (og i Norge) er det mer vanlig å bruke \mathbb{Z} for denne mengden.
- Vi lar \mathbb{R} stå for mengden av alle reelle tall.
Det er så mange reelle tall at vi vanskelig kan liste dem opp, ikke engang ved hjelp av \dots .
- Vi lar \mathbb{Q} betegne mengden av rasjonale tall.

Mengder

Det er mulig å definere \mathbb{Q} fra de andre mengdene:

$$\mathbb{Q} = \{x \in \mathbb{R} : \exists p \in \mathbb{J} \exists q \in \mathbb{N} (x = \frac{p}{q})\}.$$

I en viss forstand er det umulig å definere \mathbb{N} utfra \mathbb{R} alene.

Den forstanden faller langt utenfor pensum i MAT1030.

Mengder

Vi har allerede truffet på endel mengder i dette emnet, uten å legge vekt på at de er mengder:

Mengder

Eksempel.

- La VAL_k være mengden av fordelinger av sannhetsverdier T og F på k utsagnsvariable p_1, \dots, p_k .
- La var være en uendelig mengde $\{p_1, p_2, \dots\}$ av utsagnsvariable.
Vi kan arbeide med denne mengden uten å vite nøyaktig hva disse variablene er, det holder at vi har navn for dem.
- La $REP_{\mathbb{R}}$ være mengden av digitale representasjoner av reelle tall, og la $REP_{\mathbb{J}}$ være mengden av binære representasjoner av hele tall.
- Da vi snakket om hvordan vi skulle "forstå" kvantorer $\exists x$ og $\forall y$, sa vi at vi måtte presisere hvilke verdier x og y kan ta.
Dette kan vi gjøre mer presist ved å snakke om variasjonsmengden eller tolkningsområdet til den enkelte variable som en mengde.

Mengder

- Senere skal vi se på datastrukturer som mengder.
- En datastruktur vil bestå av de objektene en variabel i et program kan ta som verdi.
- Vi har tidligere nevnt at vi ofte må deklareere typen til en variabel.
- Dette betyr at vi må avgrense den delen av datastrukturen som den aktuelle variabelen kan hente sine verdier fra.
- For å kunne beskrive datastrukturer, datatyper og for å kunne diskutere hvordan, og i hvilket omfang, objektene i slike strukturer kan representeres som digitalisert informasjon, trenger vi et grunnlag i mengdelære.
- Det er dette grunnlaget vi skal få når vi gjennomgår stoffet i Kapittel 5.

Mengder

- Vi skal ikke kaste oss inn i den filosofiske diskusjonen om hva en mengde “er” i noen større grad enn i diskusjonen om hva et tall er for noe.
- Det viktigste for oss er å forstå hvordan vi bruker mengder til å uttrykke oss presist.
- Hvis vi senere skal diskutere teoriproblemer med en (eksempelvis) kineser, er det viktig at vi har den samme forståelsen.
- En viktig del av denne forståelsen er å vite når to mengder er like.

Mengder

Definisjon.

To mengder A og B er like hvis de har nøyaktig de samme elementene.

Det betyr at mengden er fullstendig bestemt av sine elementer, og det betyr ikke noe hvordan vi beskriver dem:

Mengder

Eksempel.

- $\{2, 3, 4\} = \{3, 4, 2\} = \{2, 2^2, 3\} = \{1 + 1, 2, 2 + 1, 3, 3 + 1, 2 + 2, 4\}$
- $\{3, 4, 7\} = \{\text{III}, \text{IV}, \text{VII}\}$ så lenge det er klart at vi bruker arabiske og romerske måter å uttrykke tall på, mens hvis vi snakker om de konkrete symbolsekvensene er mengdene forskjellige.

•

$$\{(x, y) \in \mathbb{R}^2 : 9x^2 + 16y^2 = 25\} =$$

$$\{(x, y) \in \mathbb{R}^2 : (\frac{3x}{5})^2 + (\frac{4y}{5})^2 = 1\}.$$

Mengder

Eksempel (Tre "dumme" oppgaver).

- Finn

$$\{x \in \mathbb{R} : x^4 + 4x^3 + 8x^2 + 8x + 4 = 0\}$$

- Bestem mengden av sannhetsverdifordelinger som gjør

$$(p \vee q) \wedge (\neg p \wedge \neg q)$$

sann.

- Finn mengden av flytende-punkt representasjoner av det reelle tallet 0.

- Felles for alle disse tre oppgavene er at mengdene ikke har noen elementer.
- Alle løsningsmengdene har nøyaktig de samme elementene, nemlig ingen, og er derfor like.

Mengder

Definisjon.

Vi lar \emptyset betegne den tomme mengden, det vil si mengden som ikke har noen elementer.

- Symbolet \emptyset er internasjonalt gjennomført, og man vil ikke finne det forklart andre steder enn i innføringstekster i mengdelære.
- Det er fremkommet ved en "null" med en strek over, og må ikke forveksles med noen bokstav i noe alfabet.
- Vi har at $\emptyset \neq \{\emptyset\}$; den første mengden har ingen elementer mens den andre har ett element, nemlig \emptyset selv.

Mengder

- Vi har tidligere sagt at når vi bruker en variabel i en programmeringssammenheng, må vi deklare *typen* til variabelen.
(Dette er ikke alltid sant, det kan avhenge av programmeringsspråket, men for de fleste ikke-spesialiserte språk er det tilfelle.)
- Alternativt arbeider vi med en datastruktur hvor vi henter alle variabelverdier fra.
- Vi har ofte definert mengder som $\{n \in \mathbb{N} : \dots\}$ eller $\{x \in \mathbb{R} : \dots\}$. I slike tilfeller er det ofte klart fra sammenhengen hvilke mengder n eller x skal hentes fra, og det kan være brysomt å måtte presisere det hver gang.
- Hvis A er en mengde, og vi vil se på mengden av objekter som ikke er med i A , vil vi normalt ønske å avgrense oss til de objektene som er av interesse i den aktuelle sammenhengen.

- Alt dette gjør det aktuelt å innføre et eget symbol for en universell mengde, uten at denne universelle mengden trenger å være den samme i enhver sammenheng.

Mengder

Definisjon.

- Vi lar \mathcal{E} stå for den universelle mengden.
- \mathcal{E} vil betegne forskjellige mengder i forskjellige sammenhenger, men skal ligge fast i enhver gitt sammenheng.
- I noen sammenhenger (definisjoner, oppgaver) vil \mathcal{E} bare betegne en vilkårlig universell mengde, mens i andre sammenhenger vil betydningen av \mathcal{E} bli presisert.

Mengdealgebra

- Vi skal nå se på noen grunnleggende operasjoner på mengder.
- Disse kalles Booleske operasjoner.
- Det er en nær sammenheng mellom Booleske operasjoner og utsagnslogiske bindeord.
- Utsagnslogikk og Boolesk mengdelære er begge instanser av det som kalles Boolesk algebra.
- Et mer systematisk studium av Boolesk algebra er relevant for retninger innen teoretisk informatikk.

Mengdealgebra

Eksempel.

- Vi har fått i oppgave å finne en digital representasjon av funksjonen

$$f(x) = \sqrt{x^2 - 1}$$

og vet at denne funksjonen bare er definert når $|x| \geq 1$.

- Det betyr at vi må finne mengden av 32-bit representasjoner av tall ≥ 1 og av tall ≤ -1 og så slå disse mengdene sammen.
- En slik sammenslåing kaller vi en union.

Mengdealgebra

Definisjon.

Hvis A og B er to mengder, definerer vi unionen av A og B som

$$A \cup B = \{x : x \in A \vee x \in B\}.$$

- Unionen av A og B består altså av de objektene som er element i minst en av de to mengdene A og B , men også gjerne i begge.

Mengdealgebra

Eksempel.

- \mathbb{N} er unionen av mengdene av positive partall og positive oddetall.
- Løsningsmengden til ulikheten $x^2 > 1$ er

$$\{x : x < -1\} \cup \{x : x > 1\}.$$

- Mengden av bitsekvenser som representerer tall x med $|x| \geq 1$ kan skrives som

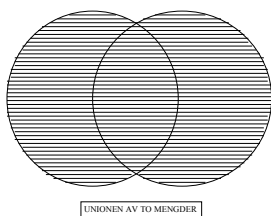
$$\{01\sigma : \sigma \in \{0, 1\}^{30}\} \cup \{11\sigma : \sigma \in \{0, 1\}^{30}\},$$

hvor $\{0, 1\}^{30}$ er en vanlig måte å uttrykke mengden av 30-bits sekvenser.

Mengdealgebra

- En vanlig måte å illustrere Booleske operasjoner på er ved å bruke et Venn-diagram
- Et Venn-diagram for en Boolesk kombinasjon av to eller tre mengder vil bestå av en sirkel for hver mengde, slik at de overlapper hverandre.
- Hvis vi har bruk for å markere den universelle mengden \mathcal{E} , gjør vi det ved et rektangel som omslutter alle sirklene.
- Ved å bruke forskjellige skraveringer, kan vi illustrere hvilke punkter som ligger i mengden og hvilke som ikke gjør det.

Mengdealgebra



- Vi har en sirkel for hver mengde.
- Vi har fire felter, et for hver kombinasjon av $x \in A$ og $x \in B$.

- Det skraverte området markerer $A \cup B$.

Mengdealgebra

Eksempel.

- La A være mengden av naturlige tall n slik at n kan deles på både 6 og 8.
- La B være mengden av reelle tall x slik at

$$((x - 1)^2 - 1)^2 + (x^4 - 16)^4 = 0$$

- La C være mengden av 8-bits representasjoner av ikke-negative partall på vår lekemas-kin.
- I tilfelle A gir vi to krav direkte.
- I tilfelle B krever vi at vi både må ha at $(x - 1)^2 - 1 = 0$ og at $x^4 - 16 = 0$.
- I tilfelle C krever vi at bitsekvensen både må starte med 0 og ende med 0.

Mengdealgebra

Definisjon.

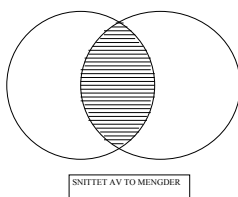
- La A og B være to mengder.
- Med snittet av A og B mener vi
-

$$A \cap B = \{x : x \in A \wedge x \in B\}.$$

- $A \cap B$ består altså av de objektene som er elementer både i A og i B .

Vi kan også illustrere snitt ved et Venn-diagram:

Mengdealgebra



ne.

Vi skraverer det feltet som markerer fellesdelen av de to mengde-

Mengdealgebra

Eksempel.

- Hvis

$$A = \{0, 1, 2, 3, 4, 5\}$$

$$B = \{1, 3, 5, 7, 9\}$$

er $A \cup B = \{0, 1, 2, 3, 4, 5, 7, 9\}$, mens $A \cap B = \{1, 3, 5\}$.

- Hvis

$$A = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 \geq 1\}$$

$$B = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 \leq 1\}$$

er $A \cup B$ hele planet, mens $A \cap B$ er mengden av punkter på enhets sirkelen.

Mengdealgebra

- Vi har ikke hatt bruk for den universelle mengden \mathcal{E} i definisjonen av union og snitt.
- Vi har formulert definisjonen av \cup og \cap slik at sammenhengen med \vee og \wedge skal komme klart frem.
- Den neste mengdeoperasjonen vi skal se på er komplement.
- Den vil ha et nært slektskap til \neg .
- For å definere komplement må vi ha tilgang til en universell mengde.

Mengdealgebra

Eksempel.

1 La P betegne mengden av primtall.

Et sammensatt tall er et naturlig tall $\neq 1$ som ikke er et primtall.

Vi kan definere

$$SAM = \{n \in \mathbb{N} : n \notin P \cup \{1\}\}.$$

Det hadde vært enklere om vi kunnet skrive dette på en kortere måte som vi alle kunne forstå.

Mengdealgebra

Eksempel (Fortsatt).

2 Vi definerer de irrasjonale tallene som mengden av de reelle tallene som ikke er rasjonale.

Dette kunne vi skrevet som

$$\{x \in \mathbb{R} : x \notin \mathbb{Q}\}.$$

Hvis alle forstår oss om vi skriver $\overline{\mathbb{Q}}$ istedenfor, ville det vært greiere.

Mengdealgebra

Eksempel (Fortsatt).

3 Vi har ord både for partall og for oddetall, men vi har ikke et eget ord for tall som ikke kan deles på 13 (eller 17 for den saks skyld).

I denne sammenhengen var det klart at vi snakker om hele tall, dvs

$$\mathcal{E} = \mathbb{J}.$$

Vi vil si at mengden av tall som ikke kan deles på 13 er komplementet til mengden av tall som kan deles på 13, og vi markerer komplementet til en mengde ved å sette en strek over uttrykket for mengden.

Mengdealgebra

Definisjon.

La \mathcal{E} være en universell mengde og la A være en mengde hvor alle elementene ligger i \mathcal{E} .

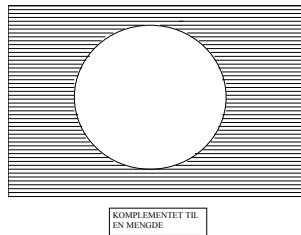
Med komplementet til A mener vi

$$\overline{A} = \{x \in \mathcal{E} : x \notin A\} = \{x \in \mathcal{E} : \neg(x \in A)\}.$$

Når vi skriver \overline{A} skal det alltid være klart at \mathcal{E} er kjent (eller at vi arbeider generelt med en vilkårlig \mathcal{E} .)

Vi kan beskrive komplementet ved et Venn-diagram.

Mengdealgebra



Det skraverte feltet markerer komplementet.

Mengdealgebra

Eksempel.

- Hvis $\mathcal{E} = \{0, 1, 2, 3, 4, 5, 6, 7\}$ og $A = \{2, 3, 5\}$, er $\bar{A} = \{0, 1, 4, 6, 7\}$.
- Hvis \mathcal{E} er som over og $B = \{1, 3, 5, 7\}$, er $\bar{B} = \{0, 2, 4, 6\}$
- Hvis $\mathcal{E} = \mathbb{J}$ og $B = \{1, 3, 5, 7\}$ er

$$\bar{B} = \{\dots, -3, -2, -1, 0, 2, 4, 6, 8, 9, 10, \dots\}.$$

Mengdealgebra

Den siste mengdeoperasjonen vi skal innføre er mengdedifferens.

Eksempel.

- Anta at vi har opprettet et hunderegister hvor vi har lagret informasjon som rase, vaksineringer og annen relevant informasjon. Foreningen for store hunder vil gjerne ha en oversikt over hvilke hunder av rasen *Grand Danois* det er som ikke har fått rabiesvaksine.

Fra basen kan vi skrive ut liste over alle hunder av rasen *Grand Danois* samt en liste over alle hunder som har fått rabiesvaksine.

Mengdealgebra

Eksempel (Fortsatt).

For at foreningen skal få den informasjonen den ønsker, må vi ta differensen mellom disse to mengdene, det vil si alle *Grand Danois* som ikke er vaksinert mot rabies.

Vi skriver $GD - R$ for denne mengden, og håper i vårt stille sinn at utvikleren av basen har lagt inn som funksjon å skrive ut differens mellom to mengder hentet fra basen.

Mengdealgebra

Definisjon.

Hvis A og B er to mengder, er differansen A minus B definert ved

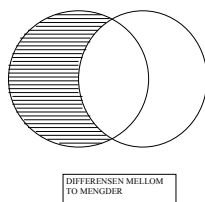
$$A - B = \{x : x \in A \wedge x \notin B\}$$

Vi bruker ofte betegnelsen mengdedifferens

En alternativ skrivemåte mye brukt i litteraturen er $A \setminus B$.

Vi kan også illustrere mengdedifferens ved et Venn-diagram.

Mengdealgebra



- Det skraverte området markerer differensen
- Vi har ikke hatt bruk for ε her.
- $A - B = A \cap \bar{B}$

Venn-diagrammer

- Den virkelige nytten av Venn-diagrammer ligger i at de kan brukes til å studere sammenhengen mellom forskjellige Booleske uttrykk.
- I en viss forstand er bruk av Venn-diagrammer en parallell til bruk av sannhetsverditabeller.
- Vi skal vise noen eksempler på tavlen på hvordan vi kan etablere mengdeteoretiske identiteter ved hjelp av Venn-diagrammer.
- Et slikt eksempel ble gjennomgått 11/2, resten ble utsatt til 13/2.

Forelesning 10

Mengdelære

Dag Normann - 13. februar 2008

Venn-diagrammer

- Mandag innførte vi de Booleske operasjonene
 - Union \cup
 - Snitt \cap
 - Komplement \bar{A}
 - Mengdedifferens $A - B$samt de faste mengdene \emptyset og \mathcal{E} .

Venn-diagrammer

- Vi tegnet Venn-diagrammet tilhørende de forskjellige Booleske operasjonene, og begynte på eksempler på litt mer avansert bruk av Venn-diagrammer.
- Dette skal vi fortsette med nå, fortsatt på tavlen.

Venn-diagrammer

Eksempel.

- deMorgans lover
 - $\overline{A \cup B} = \bar{A} \cap \bar{B}$
 - $\overline{A \cap B} = \bar{A} \cup \bar{B}$
- De distributive lovene
 - $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
 - $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

Venn-diagrammer

Eksempel (Fortsatt).

- $A \cap (B \cup C) = (A - B) \cap (A - C)$
- $(\bar{A} - B) \cap C = C - (A \cup B)$

Venn-diagrammer

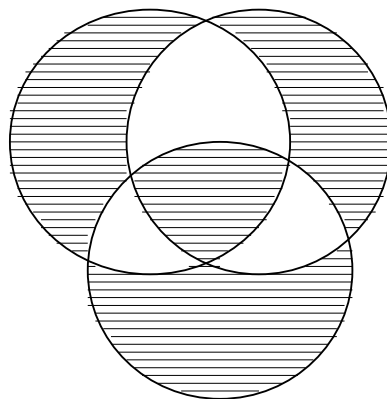
Oppgave.

Vi definerer ofte symmetrisk differens ved

$$A \triangle B = (A - B) \cup (B - A).$$

- Illustrer $A \triangle B$ ved et Venn-diagram.
- Vis at $(A \triangle B) \triangle C$ kan illustreres ved Venn-diagrammet på neste side.
- Drøft hvorfor dette viser at vi kunne skrevet $A \triangle B \triangle C$ uten bruk av parenteser.

Oppgaveillustrasjon



Venn-diagrammer

Oppgave.

- Vi bruker bare Venn-diagrammer for uttrykk med en, to eller tre mengder.
- Tegn et Venn-diagram for tre mengder A , B og C , og sett inn sannhetsverdiene for de tre basisutsagnene $x \in A$, $x \in B$ og $x \in C$ i de forskjellige feltene.
- Undersøk hvor mange deler det er mulig å dele planet inn i ved hjelp av fire sirkler.
- Forklar hvorfor dette viser at Venn-diagrammer ikke er hensiktsmessige for Booleske uttrykk med mer enn tre mengder.

Inklusjon

Eksempel.

- Det er selvfølgelig slik at alle tall som kan deles på 4 også er partall.
Vi sier da at mengden av tall delelige med 4 er inneholdt i partallene, eller at den er en delmengde av partallene.
- Mengden av registrerte fødselsnummere er inneholdt i mengden av alle data registrert i skattedirektoratet.
- Mengden av hunder forsikret i et forsikringselskap er en delmengde av mengden av dyr forsikret i selskapet.
Dette er igjen en delmengde av mengden av objekter (dyr, boliger, biler m.m.) som er forsikret i selskapet.

Inklusjon

Definisjon.

Hvis A og B er mengder, sier vi at A er inneholdt i B , eller at A er en delmengde av B , hvis

$$\forall x(x \in A \rightarrow x \in B).$$

- Vi skriver

$$A \subseteq B$$

for A er inneholdt i B .

Inklusjon

- Vi vil kunne skrive $A \subseteq B$ selv om $A = B$.
- Noen forfattere bruker $A \subset B$ slik vi bruker $A \subseteq B$ mens andre bruker det i betydningen

$$A \subseteq B \wedge A \neq B.$$

- I dette siste tilfellet vil vi si at A er ekte inneholdt i B .

Inklusjon

Eksempel.

- $\{2, 5, 6\} \subseteq \{1, 2, 5, 6, 7\}$ og inklusjonen er ekte.
- I følge læreboka vil

$$\mathbb{N} \subseteq \mathbb{J} \subseteq \mathbb{Q} \subseteq \mathbb{R}.$$

Når vi ser på disse mengdene som datatyper, vet vi at vi må bruke forskjellige måter å representere et tall i \mathbb{J} på, avhengig av om vi ser på tallet som et element i \mathbb{J} eller \mathbb{R} .

Denne påstanden er derfor ikke helt uproblematisk, men dog akseptabel for våre formål.

- $\{x : x^2 > 4\} \subseteq \{x : x^2 > 4 \vee x < -1\}$.

Inklusjon

Vi kan bruke Venn-diagrammer til å vise at et Boolesk uttrykk alltid definerer en delmengde av mengden definert ved et annet Boolesk uttrykk:

Eksempel.

- $A \cap B \subseteq A \cup B$
- $\bar{A} \cap (B - C) \subseteq B - (A \cap C)$

- *I en konkret situasjon kan vi ha inklusjon selv om ikke Venn-diagrammet viser det.*

Boolesk algebra

- Det er en nær sammenheng mellom Boolesk mengdealgebra og utsagnslogikk.
- Ved å erstatte A med $x \in A$ oppfattet som en utsagnsvariabel, kan vi spisse \cup til \vee , \cap til \wedge og erstatte komplement \bar{A} med $\neg(x \in A)$, og vi får en utsagnslogisk formel.
- Det er da naturlig å erstatte \emptyset med F og \mathcal{E} med T.
- To mengder vil alltid være like nøyaktig når oversettelsene er logisk ekvivalente.

Boolesk algebra

- Tabell 5.1 på side 79 i læreboka lister noen Booleske identiteter.
- De har sine paralleller i tabellen på side 55 over logikkens lover.
- Vi skal ikke drille regning med disse Booleske identitetene, men noen av dere vil kunne møte dem igjen i senere emner.

En digresjon

- Hvis vi hadde kunnet snakke om mengden av alle mengder, hadde vi hatt en grunn mindre til å bringe inn den universelle mengden \mathcal{E} .
- Antagelsen om at det finnes en mengde som har alle mengder som elementer, leder imidlertid til en motsigelse som kalles Russels Paradoks.
- Vi gir beviset for Russels Paradoks som en oppgave med hint.

Russels Paradoks

Oppgave.

- Anta at X er en mengde, og at for alle mengder Y vil $Y \in X$.
- La $Z = \{Y \in X : Y \notin Y\}$.
- Da er $Z \in X$.
- Vis at hvis $Z \in Z$ vil $Z \notin Z$.
- Vis at hvis $Z \notin Z$ vil $Z \in Z$.
- Forklar hvorfor dette viser at mengden X ikke kan finnes.

Digital representasjon av mengder

- I utgangspunktet skal det ikke spille noen rolle i hvilken rekkefølge man skriver opp elementene i en mengde.
- Hvis man imidlertid har behov for å representere visse mengder digitalt, må man velge seg en rekkefølge på elementene i den universelle mengden \mathcal{E} .
- Vi skal nå se på en metode for digital representasjon av mengder som virker når \mathcal{E} er endelig.
- Hvis \mathcal{E} er en uendelig mengde, må man enten velge en annen metode eller gi opp.

Digital representasjon av mengder

Definisjon.

- Anta at \mathcal{E} har k elementer i rekkefølge

$$\{a_1, \dots, a_k\}.$$

- La $A \subseteq \mathcal{E}$
- Vi representerer A som informasjon på k bits i rekkefølge, ved at bit nummer i får verdien 1 hvis og bare hvis $a_i \in A$.

Digital representasjon av mengder

- Ved denne måten å representere mengder på blir det svært enkelt å etterlikne de Booleske operasjonene.
- Snitt svarer til punktvis multiplikasjon, union svarer til det å ta maksimumsverdien punktvis og komplement svarer til å skifte verdi i alle bits.
- Vi kommer ikke til å jobbe med digital representasjon av mengder på senere forelesninger.
- Digital representasjon av mengder inngår imidlertid i en av oppgavene i første oblig. sett.

Kardinaltall

- Hvis vi i noen sammenhenger ønsker å bruke digitale representasjoner av mengder, er det viktig at ε ikke får lov til å være for stor.
- $10^{10^{10}}$ er lett å skrive, men foreløpig har vi ingen datamaskin med så mange bits.
- For å kunne følge med på hvor store mengder vi opererer med, og for å kunne resonere omkring størrelse på mengder, er det en fordel med en notasjon for størrelsen av mengder.
- Det er dette vi vil fange opp i begrepet kardinaltall.

Kardinaltall

Definisjon.

- La A være en endelig mengde.
- Med kardinaltallet til A mener vi antall elementer i A .
- Vi skriver $|A|$ for kardinaltallet til A .

Kardinaltall

- Den tyske matematikeren *Georg Cantor* utviklet en teori for kardinaltallet til en uendelig mengde også.
- Dette skjedde i siste halvdel av 1800-tallet.
- Ut fra Cantors definisjon finnes det like mange rasjonale tall og hele tall som naturlige tall, mens det finnes ekte flere reelle tall.
- Vi skal begrense oss til kardinalitet av endelige mengder.
- Selv om datamaskiner av natur bare kan håndtere endelig mye informasjon, har imidlertid studiet av uendelige mengder også en plass i informatikken.

Kardinaltall

Eksempel.

a) La $A = \{0, 1, 2\}$.

Da har A 8 delmengder: $\emptyset, \{0\}, \{1\}, \{2\}, \{0, 2\}, \{0, 1\}, \{1, 2\}$ og $\{0, 1, 2\}$.

Disse er skrevet opp i en usystematisk rekkefølge.

En mer systematisk måte vil være først å skrive den ene delmengden av \emptyset : \emptyset ,

så resten av delmengdene av $\{0\}$: $\{0\}$

så resten av delmengdene av $\{0, 1\}$: $\{1\}$ og $\{0, 1\}$

og til slutt resten av delmengdene av $\{0, 1, 2\}$: $\{2\}, \{0, 2\}, \{1, 2\}$ og $\{0, 1, 2\}$

Kardinaltall

Eksempel (Fortsatt).

Den naturlige rekkefølgen blir da

$$\{\emptyset, \{0\}, \{1\}, \{0, 1\}, \{2\}, \{0, 2\}, \{1, 2\}, \{0, 1, 2\}\}$$

- b) For å liste opp alle delmengder av $\{0, 1, 2, 3\}$ lister vi først opp alle delmengder av $\{0, 1, 2\}$ og deretter alle nye delmengder, ved å legge 3 til en av de åtte første.

Det gir

$$\{\emptyset, \{0\}, \{1\}, \{0, 1\}, \{2\}, \{0, 2\}, \{1, 2\}, \{0, 1, 2\},$$

$$\{3\}, \{0, 3\}, \{1, 3\}, \{0, 1, 3\}, \{2, 3\}, \{0, 2, 3\}, \{1, 2, 3\}, \{0, 1, 2, 3\}\}$$

Potensmengder

Definisjon.

- La A være en mengde.
- Med potensmengden til A mener vi mengden av alle delmengder av A .

Merk.

- Hvis A er en mengde og $B \subseteq A$ er en vilkårlig delmengde, vil vi for hver $x \in A$ ha to muligheter, $x \in B$ og $x \notin B$.
- En konsekvens er at hvis A er endelig vil potensmengden til A ha $2^{|A|}$ elementer.
- Dette vil ofte bety at det vil ta alt for lang tid å gjennomføre naive algoritmer.

Potensmengder

Eksempel.

- La A være en endelig mengde av naturlige tall.
- Vi lar $\sum A$ bety summen av alle tallene i A .
- Partisjonsproblemet er om det finnes delmengder B og C av A som er slik at
 1. $A = B \cup C$
 2. $\emptyset = B \cap C$ (De er disjunkte)
 3. $\sum B = \sum C$

- Den første strategien kan være å liste opp alle par $B \subseteq A$ og $C = A - B$, og sjekke. men hvis A har 1000 elementer, er ikke dette praktisk gjennomførbart.
- Ingen vet pr. i dag om det finnes en vesentlig raskere metode til å løse partisjonsproblemet generelt.
- Partisjonsproblemet er et eksempel på et NP-komplett problem.

Potensmengder

Merk.

- Potensmengden til A er definert selv om A er uendelig.
- I det tilfellet er ikke alle egenskapene ved potensmengder fullstendig klarlagt ennå.
- Vi ledes langt ut over rammene for diskret matematikk om vi prøver å forstå potensmengden til en uendelig mengde.
- Cantor viste at i en viss forstand er potensmengden til A alltid ekte større enn A .

Ordnete par

- Vi har brukt mengden \mathbb{R}^2 av tallpar i tidligere eksempler.
- Alle vet at det er forskjell på tallparene $(2, 3)$ og $(3, 2)$ i \mathbb{R}^2 .
- Det betyr at rekkefølgen på tallene i paret spiller en rolle.
- Et slikt par kaller vi et ordnet par.

Ordnete par

- Det er ikke bare tall som kan opptre i par.
- Vi kan for eksempel skrive at

Per og Kari er ektefeller

og vi mener så absolutt at de utgjør et par.

- I dette tilfellet betyr ikke rekkefølgen noe, men skriver vi

Kari er kona til Per

kan vi ikke erstatte det med

Per er kona til Kari.

Ordnete par

- Vi trenger begrepet ordnet par for å kunne snakke presist og generelt om visse former for sammenhenger vi kan finne mellom to objekter.
- Disse objektene kan være tall i en tallmengde.
- De kan imidlertid også være data i en base, data som representerer personer, hendelser, adresser, yrker og mye annet det kan være behov for å registrere.

- Derfor vil vi legge en helt generell definisjon til grunn, når vi definerer hva som menes med et ordnet par.

Ordnete par

Definisjon.

La a og b være to objekter.

Det ordnede paret (a, b) av a og b er a og b skrevet i rekkefølge.

To ordnede par (a, b) og (c, d) er like hvis $a = c$ og $b = d$.

Merk.

- Vi har egentlig ikke sagt hva et ordnet par er for noe, bare knyttet det til at objektene settes i rekkefølge.
- Det er definisjonen av når to ordnede par er like som gir oss den ønskede matematiske presisjonen. Det knytter den abstrakte definisjonen opp til skrivemåten vi benytter.

Ordnete par

Definisjon.

La A og B være to mengder.

Med det Cartesiske produktet $A \times B$ av A og B mener vi

$$A \times B = \{(a, b) : a \in A \wedge b \in B\}.$$

Betegnelsen henter sitt navn fra den franske matematikeren René Descartes, eller Renatus Cartesius som var det latinske navnet han tok seg.

Ordnete par

Eksempel.

- La A være mengden av registrerte norske skøyteløpere og B være mengden av tider mellom 1.30.00 og 2.30.00.
Da vil registreringer av personlige rekorder på 1500m oppfattes som par i $A \times B$

- Hvis A er mengden av ord skrevet med latinske bokstaver og B er antall sider på nettet, leter vi i prinsippet gjennom $A \times B$ når vi søker etter nettsider hvor et bestemt ord forekommer.

I dette tilfellet er det klart at vi trenger å utvikle spesielle teknikker for å kunne gjøre dette på en effektiv måte, men utviklingen av slike teknikker starter med å forstå kompleksiteten av $A \times B$.

Ordnete par

Hvis A og B er endelige mengder, vil

$$|A \times B| = |A| \cdot |B|.$$

For de som har lært om matriser, ser vi sammenhengen med en $n \times m$ -matrise.

La $A = \{a_1, \dots, a_n\}$ og $B = \{b_1, \dots, b_m\}$.

Da kan vi skrive $A \times B$ som:

Ordnete par

$$\left\{ \begin{array}{ccc} (a_1, b_1) & \cdots & (a_1, b_m) \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ (a_n, b_1) & \cdots & (a_n, b_m) \end{array} \right\}$$

Ordnete par

Eksempel (Fortsatt fra side 104).

- Hvis A er mengden av norske statsborgere, vil $A \times A$ være mengden av par av norske statsborgere.

Det finnes mange interessante undermengder av $A \times A$ bestemt av de forskjellige forhold det kan være mellom to personer, eksempelvis

- kollega av
- søster til
- nabo av
- misunnelig på
- ...

Dette vil lede oss over til avsnittet om relasjoner.

Relasjoner

Definisjon.

La A være en mengde.

En binær relasjon på A er en delmengde R av $A^2 = A \times A$.

Merk.

- I senere studier kan dere komme borti relasjoner mellom tre eller flere objekter. Disse er da ikke binære.
- Siden vi bare skal studere binære relasjoner, gjør vi som boken, og dropper ordet "binær".

Relasjoner

Eksempel.

- La A være mengden av hele tall vi representerer som tidligere beskrevet i en datamaskin. La $(a, b) \in R$ hvis $a < b$ mens $(a, b) \in S$ om representasjonen av a er et mindre binært tall enn representasjonen av b . Begge relasjonene kan brukes hvis vi skal søke etter et eksempel eller et moteksempel, men søkene kan gi forskjellige resultater. Det er lettest å programmere en gjennomsøking av mengden hvis vi bruker ordningen S .

Relasjoner

Eksempel (Fortsatt).

- I kryptografi er moduloeregning viktig. Hvis p er et primtall og a og b er hele tall, sier vi at $a \equiv_p b$ om p er en faktor i $a - b$. Vi kunne like gjerne skrevet

$$(a, b) \in \equiv_p .$$

Relasjonene \equiv_p og beslektede relasjoner (hvor eksempelvis p ikke er et primtall, men i praksis umulig å faktorisere) spiller en stor rolle i arbeidet for sikker overføring av sensitive data.

Relasjoner

Eksempel.

- Kompliserte prosesser kan gjerne beskrives ved hjelp av et flyt-diagram.
- Hvert ledd i prosessen blir beskrevet ved en node, og de mulige utviklingene i prosessen blir beskrevet ved piler.
- Vi skal ikke gi noen innføring i flytdiagrammer men se på et enkelt eksempel:
- En sjokoladeautomat kan ta imot 20-kroner, kan gi ut sjokolader til verdier av 10 kroner og 20 kroner, og kan gi vekslepenger.
- Automaten illustreres på tavlen
- Den relevante relasjonen er den som markeres med pilene.

Relasjoner

- Noen lærebøker vil definere en relasjon fra A til B som en mengde

$$R \subseteq A \times B.$$

- Det kan finnes pedagogiske grunner for å gjøre det slik, men enhver relasjon fra A til B vil samtidig være en relasjon på $A \cup B$.

Relasjoner

- Det å beskrive en relasjon som en mengde av ordnede par gir ikke mye innsikt i hvordan relasjonen ser ut.
- Det å skrive $(a, b) \in R$ representerer også en uvant måte å skrive ting på.
- Ingen av oss har lyst til å begynne å skrive $(2, 3) \in <$ i stedet for $2 < 3$ eller $(3, 3) \in =$ i stedet for $3 = 3$,
for ikke å snakke om $(\emptyset, \{\emptyset\}) \in \in$ i stedet for $\emptyset \in \{\emptyset\}$.
(Skulle vi finne på noe slikt, ville vi rote oss bort i grunnlagsproblemer som langt overstiger det vi skal ta opp i MAT1030, samt gjøre noe helt unyttig.)
- Den første forenklingen vi skal gjøre er å skrive aRb når vi mener $(a, b) \in R$.

Relasjoner

- Hvis A er en relativt liten mengde, finnes det to måter å beskrive R på,
 - Ved hjelp av en matrise
 - Ved hjelp av en graf
- Vi skal se på noen eksempler.
- For begge måter å beskrive relasjoner på spiller det en stor rolle hvordan man organiserer elementene i mengden A,
 - i rekkefølge som koordinater
 - som punkter på en tavle eller et ark

Relasjoner

- La $A = \{1, 2, 3, 4, 5\}$
- La $R = \{(1, 3), (2, 4), (3, 5), (4, 1), (5, 2)\}$
- Vi vil illustrere R ved hjelp av en 5×5 -matrise.
- Radene, regnet ovenfra, vil representere 1. koordinat.
- Søylene, regnet fra venstre, vil representere 2. koordinat.
- Vi markerer elementene i R med T og de parene som ikke er med i R med F.
- Det er like vanlig, og av forskjellige grunner bedre, å bruke 1 og 0.

Relasjoner

$$\begin{bmatrix} F & F & T & F & F \\ F & F & F & T & F \\ F & F & F & F & T \\ T & F & F & F & F \\ F & T & F & F & F \end{bmatrix}$$

Det hadde vært lettere om man hadde brukt farver, eller 1 og 0 for å se hvor "pen" denne relasjonen er.

Den grafiske fremstillingen tar vi på tavlen.

Relasjoner

La $A = \{1, 2, 3, 4\}$ og $R = \{(1, 2), (1, 3), (1, 4), (4, 1), (3, 1), (2, 1), (3, 3)\}$

Matriseformen blir

$$\begin{bmatrix} F & T & T & T \\ T & F & F & F \\ T & F & T & F \\ T & F & F & F \end{bmatrix}$$

- Den grafiske fremstillingen tar vi på tavlen.

Forelesning 11

Relasjoner

Dag Normann - 18. februar 2008

Oppsummering

- Vi har gjort oss ferdige med innføringen av Boolesk mengdelære.
- Bruk av Venn-diagrammer er et av læringsmålene i dette emnet.
- Vi så kort på digital representasjon av mengder.
- Forståelsen av dette blir testet i Oblig. 1
- Definisjonene av kardinaltall og potensmengde, og sammenhengene mellom dem, bør dere kjenne til.
- Vi startet med definisjonene av ordnede par, Cartesiske produkter og relasjoner.
- Dette skal vi fortsette med.

Relasjoner

- Erfaringsmessig faller det stoffet vi nå begynner på vanskeligere for mange enn det vi har gått gjennom til nå.
- Det skyldes at stoffet synes abstrakt og at det er mange nye begreper.
- I eksempler vil vi kunne innføre noen nye symboler som vi gir en spesiell betydning.
- Et eksempel fra forrige uke er relasjonen $a \equiv_p b$ som uttrykker at p er en faktor i $a - b$.
- En av de ferdighetene vi skal oppøve i MAT1030 er evnen til å lese, og forstå, definisjoner.
- De relasjonene vi vil innføre i eksemplene, vil som oftest ikke være pensum, men evnen til å forstå slike eksempler kan bli prøvet til eksamen.

Relasjoner

- Relasjoner med bestemte egenskaper kan dukke opp i så mange forskjellige sammenhenger at det kan være aktuelt å studere dem under ett.
- Det kan også være aktuelt å utvikle program som virker for alle relasjoner av en gitt type.
- Følgende oppgaver trenger strengt tatt det samme programmet:
 - Ordne dagens avisoverskrifter alfabetisk.
 - Ordne deltagerne i kretsmesterskapet på 10 km klassisk etter oppnådd slutt-tid.
 - Ordne LOTTO-tallene etter størrelse.
- Det er noe felles ved oppgaven å skulle ordne en mengde, og det er naturlig å isolere de relasjonene som kan oppfattes som ordninger.

Relasjoner

- Vi ordner gjerne studentene etter oppnådde karakterer.
Er det noen forskjell på en slik "ordning" og en alfabetisk ordning?
- Inklusjon mellom mengder er en form for "større enn", men det finnes mengder, eksempelvis $\{1, 2, 3\}$ og $\{2, 3, 4\}$, som ikke er inneholdt i hverandre noen vei.
- Hva vil sorteringsalgoritmer gjøre hvis vi bruker dem på slike ordninger?

Relasjoner

- Det er noen egenskaper ved relasjoner som er så vanlig forekommende (hos de *nyttige* relasjonene) at vi har gitt dem egne navn.
- Vi gir listen først og drøfter hver enkelt egenskap etterpå:

Relasjoner

Definisjon.

- En relasjon R på en mengde A kalles:
 - Refleksiv hvis xRx for alle $x \in A$.
 - Irrefleksiv hvis vi ikke har noen $x \in A$ slik at xRx .
 - Symmetrisk hvis xRy medfører yRx for alle $x, y \in A$.
 - Antisymmetrisk hvis $xRy \wedge yRx$ medfører at $x = y$ for alle $x, y \in A$.
 - Transitiv hvis $xRy \wedge yRz$ medfører xRz for alle $x, y, z \in A$.
- Vi skal bruke den tiden vi trenger til å lære oss disse begrepene, og å forstå dem.

Refleksive relasjoner

Eksempel (Refleksiv: xRx for alle x).

- For enhver mengde A vil likhetsrelasjonen $x = y$ være refleksiv på A .
- Hvis X er mengden av delmengder av en mengde \mathcal{E} , vil inklusjonsrelasjonen $A \subseteq B$ være refleksiv på X .
- \leq er en refleksiv relasjon, uansett om vi ser på den som en relasjon på \mathbb{N} , på \mathbb{Q} , på \mathbb{J} eller på \mathbb{R} .
- $<$ er normalt ikke en refleksiv relasjon, spesielt ikke nå vi bruker tegnet på vanlig måte for \mathbb{N} etc.
- Hvis A er mengden av sammensatte utsagn i utsagnsvariablene p, q og r , og ϕ og ψ er sammensatte utsagn, kan vi definere $\phi R \psi$ som

$$\phi \rightarrow \psi \text{ er en tautologi.}$$

Da er R en refleksiv relasjon.

Refleksive relasjoner

Hvis en relasjon gis på matrisform, er det lett å se om relasjonen er refleksiv eller ikke:

$$\begin{bmatrix} T & T & F & F & F \\ T & T & F & F & F \\ T & F & T & F & T \\ T & T & F & T & F \\ T & F & F & F & T \end{bmatrix}$$

Ved å se at vi har bare T på diagonalen, ser vi at relasjonen er refleksiv.

Gjør vi en liten forandring, trenger ikke relasjonen lenger å være refleksiv:

Refleksive relasjoner

$$\begin{bmatrix} T & T & F & F & F \\ T & F & F & F & F \\ T & F & T & F & T \\ T & T & F & T & F \\ T & F & F & F & T \end{bmatrix}$$

Irrefleksive relasjoner

Eksempel (Irrefleksiv: $\neg(xRx)$ for alle x).

- \neq er irrefleksiv på alle mengder.
- *Far til* og *Mor til* er irrefleksive relasjoner på enhver forsamling av mennesker.
- $<$ og $>$ er irrefleksive relasjoner på \mathbb{N} , \mathbb{J} , \mathbb{Q} og \mathbb{R} .
- *Ekte inklusjon* er en irrefleksiv relasjon.

Irrefleksive relasjoner

Eksempel.

- La X være mengden av sammensatte utsagn i utsagnsvariablene p , q og r , la ϕ og ψ være sammensatte utsagn, og definer relasjonen S ved

$$\phi S \psi$$

når

$$\phi \wedge \psi \rightarrow F$$

er en tautologi.

- Da er S hverken refleksiv eller irrefleksiv.

Irrefleksive relasjoner

Hvis relasjonen blir beskrevet ved hjelp av en matrise, er det også enkelt å kontrollere om relasjonen er irrefleksiv:

$$\begin{bmatrix} F & T & F & F & F \\ T & F & F & F & F \\ T & F & F & F & T \\ T & T & F & F & F \\ T & F & F & F & F \end{bmatrix}$$

Det er bare å sjekke om det står F langs diagonalen.

Symmetriske relasjoner

Eksempel (Symmetrisk: xRy medfører yRx for alle x og y).

- Symmetriske relasjoner er de relasjonene hvor rekkefølgen ikke spiller noen rolle.
- x er gift med y .
- $\phi \wedge \psi$ er ikke en kontradiksjon.
- $\phi \wedge \psi$ er en kontradiksjon.
- n og m har en felles faktor > 1 , som en relasjon på \mathbb{N} .
- $A \cap B = \emptyset$ som relasjon på potensmengden til \mathcal{E} .

Symmetriske relasjoner

Vi kan undersøke om en relasjon er symmetrisk ved å studere matriserepresentasjonen:

Eksempel.

$$\begin{bmatrix} T & T & T & F \\ T & F & F & T \\ T & F & T & T \\ F & T & T & F \end{bmatrix}$$

Denne relasjonen er symmetrisk

Eksempel.

$$\begin{bmatrix} T & F & T & F \\ T & F & F & T \\ T & F & T & T \\ F & T & T & F \end{bmatrix}$$

Denne relasjonen er *ikke* symmetrisk

Antisymmetriske relasjoner

Eksempel (Antisymmetrisk: $xRy \wedge yRx$ medfører $x = y$).

- I en antisymmetrisk relasjon skal vi ikke ha andre positive speilsymmetrier om diagonalen enn de som ligger på diagonalen.
- Inklusjon av mengder.
- \leq og $<$ i vanlige sammenhenger.
- Antisymmetri er svært ofte knyttet til former for ordninger, og vi kommer tilbake til dette senere

Transitive relasjoner

Eksempel (Transitiv: $xRy \wedge yRz$ medfører xRz).

- $<$ og \leq er transitive relasjoner i alle vanlige sammenhenger.
- \subseteq er en transitiv relasjon på potensmengden til \mathcal{E} .
- " ϕ er en logisk konsekvens av ψ " er transitiv.
- *Far til* og *Mor til* er ikke transitive, men *etterkommer* er transitiv.
- *Søsken til* er transitiv hvis vi mener helsøsken, men ikke hvis vi inkluderer halvsøsken.

Noen eksempler

- Vi skal ta utgangspunkt i noen relasjoner som det kan være aktuelt å studere av praktiske eller teoretiske grunner.
- Vi skal se på hvilke av de fem egenskapene vi har sett på disse relasjonene vil ha.

Noen eksempler

- Hvis A og B er utsagn i predikatlogikk, lar vi $A \Rightarrow B$ bety at B er en logisk konsekvens av A. Vi sier at A impliserer B.

- Det betyr at B er sann i enhver situasjon som gjør A sann.
- Vi oppfatter \Rightarrow som en relasjon på mengden av utsagn i predikatlogikk.
- \Rightarrow er *refleksiv* fordi $A \Rightarrow A$ for alle utsagn A.
- \Rightarrow er da ikke *irrefleksiv*.
- \Rightarrow er ikke *symmetrisk* (se 1.) eller *antisymmetrisk* (se 2.)
 1. Vi har $x > 0 \Rightarrow x > 0 \vee x = 0$, men omvendingen gjelder ikke.
 2. Utsagnene $\neg(x < 0) \vee x > 0$ og $x < 0 \rightarrow x > 0$ impliserer hverandre, men de er ikke like.
- \Rightarrow er transitiv siden $A \Rightarrow C$ når $A \Rightarrow B$ og $B \Rightarrow C$.

Noen eksempler

- Når man skal gi en matematisk beskrivelse av hva et program P “gjør”, er det ofte to relasjoner som det er aktuelle å studere.
- Vi begrenser oss til språk som minner om pseudokoder.
- Da har vi variable x_1, \dots, x_n i programmet.
- Underveis vil verdiene på disse variablene endre seg, gjennom instruksjoner som

$$x_i \leftarrow t(x_1, \dots, x_n).$$

- En fordeling av verdier på variablene kaller vi en valuasjon eller en tilstand.

Noen eksempler

- La V være mengden av valuasjoner, og la u og v være elementer i V .
- Hvert program \mathcal{P} vil bestemme en relasjon $\llbracket \mathcal{P} \rrbracket$ på V , hvor

$$u \llbracket \mathcal{P} \rrbracket v$$

hvis output-valuasjonen er v når inputvaluasjonen er u .

Noen eksempler

- Hvis $\llbracket \mathcal{P} \rrbracket$ er *refleksiv*, betyr det at programmet i realiteten lar alle inputvaluasjoner være uforandret.
- Hvis $\llbracket \mathcal{P} \rrbracket$ er *irrefleksiv*, betyr det at programmet gjør endringer uansett input.
- Hvis $\llbracket \mathcal{P} \rrbracket$ er symmetrisk, betyr det at hvis vi kjører programmet en gang til, kommer vi tilbake til utgangspunktet.
- Krypteringsmaskinen ENIGMA brukt av tyskerne under krigen hadde den egenskapen.
- Hvis $\llbracket \mathcal{P} \rrbracket$ er antisymmetrisk, betyr det at vi aldri kommer tilbake til input-dataene ved å kjøre programmet på output-dataene.
- $\llbracket \mathcal{P} \rrbracket$ er i praksis aldri transitiv, siden dette ville medført at vi oppnår det samme om vi kjører programmet to ganger, hvor output overføres til input mellom gangene. Dette vil imidlertid være tilfelle om output-dataene alltid inneholder en form for stopp-ordre.

Noen eksempler

- En annen viktig relasjon i studiet av hva programmer "gjør" er \vdash^*
- Et program P er normalt en form for tekst, og den teksten består av punkter eller instruksjoner.
- Typisk har vi nummerert alle linjene i en pseudokode, slik at vi kan snakke om at vi er i en bestemt posisjon i programmet underveis i kjøringen av programmet.
- La P være mengden av posisjoner og la fortsatt V være mengden av valuasjoner.
- \vdash^* er relasjonen på $P \times V$ hvor

$$(p, v) \vdash^* (q, u)$$

hvis programmet \mathcal{P} , om vi er i posisjon p og med valuasjon v vil komme til posisjon q og med valuasjon u etter ingen, ett eller flere regneskritt.

- Denne relasjonen er selvfølgelig avhengig av \mathcal{P} , og vi kan skrive den $\vdash_{\mathcal{P}}^*$.

Noen eksempler

- Denne relasjonen er *refleksiv*, fordi vi tillater at vi ikke tar noen regneskritt. Den er da normalt ikke irrefleksiv.
- Denne relasjonen er *transitiv*.
- Hvis relasjonen er symmetrisk, er det en katastrofe for programmereren, siden vi da bare har løkkeberegninger.
- Hvis relasjonen er *antisymmetrisk*, vil beregningen aldri gå i løkke.

Ekvivalensrelasjoner

En viktig klasse av relasjoner er ekvivalensrelasjonene. La oss se på noen eksempler før vi gir den formelle definisjonen.

Eksempel.

a) La $A = \{0, \dots, 9\}$.

Hvis a og b er elementer i A lar vi aRb hvis $a - b$ er delelig med 3.

Vi ser at R deler A opp i tre disjunkte mengder $B = \{0, 3, 6, 9\}$, $C = \{1, 4, 7\}$ og $D = \{2, 5, 8\}$, hvor hver mengde består av tall som står i innbyrdes relasjon til hverandre, mens vi ikke har noen R -forbindelser på mengdene imellom.

Ekvivalensrelasjoner

Eksempel (Fortsatt).

b) La $A = \mathbb{R}^2$ og la

$$(x, y)R(u, v) \Leftrightarrow x^2 + y^2 = u^2 + v^2.$$

Vi ser at to punkter står i R-relasjon til hverandre nøyaktig når avstanden til origo er den samme.

Denne relasjonen deler \mathbb{R}^2 opp i uendelig mange disjunkte mengder, nemlig sirklene om origo med radius r for $r \geq 0$.

Ekvivalensrelasjoner

Eksempel (Fortsatt).

c) La $A = \mathbb{Q}$ og la pRq om p og q har den samme *heltallsverdien*.

Heltallsverdien til et tall p er det største hele tallet $a \leq p$.

I alle disse eksemplene har vi definert en relasjon R ved at to objekter står i relasjon til hverandre når de deler en viss felles egenskap.

Det er denne typen relasjoner vi vil kalle *ekvivalensrelasjoner*.

Ekvivalensrelasjoner

- Uansett hvilken egenskap det vil være snakk om, vil ethvert objekt dele denne egenskapen med seg selv.
Vi vil altså kreve av en ekvivalensrelasjon at den skal være *refleksiv*.
- Hvis a og b deler en egenskap, kan vi også snu på ordstillingen og si at b og a deler denne egenskapen.
- Det er altså et rimelig krav til at en relasjon skal kalles en ekvivalensrelasjon at den er *symmetrisk*.
- Hvis a og b deler en egenskap, og b og c deler den samme egenskapen, er den også felles for a og c .

Vi vil kreve av en ekvivalensrelasjon at den skal være *transitiv*.

Ekvivalensrelasjoner

- Det viser seg at disse tre egenskapene er tilstrekkelige til å fange opp intuitisjonen vår om å formalisere det uformelle "dele visse egenskaper".

Den formelle definisjonen er:

Definisjon.

La R være en relasjon på en mengde A .

Vi kaller R en ekvivalensrelasjon om R er refleksiv, symmetrisk og transitiv.

Forelesning 12

Relasjoner, Funksjoner

Dag Normann - 20. februar 2008

Oppsummering

- En relasjon på en mengde A er en delmengde $R \subseteq A \times A = A^2$.
- Vi har satt navn på visse egenskaper relasjoner som oppstår i anvendelser ofte kan ha.

Definisjon.

- En relasjon R på en mengde A kalles:
 - Refleksiv hvis xRx for alle $x \in A$.
 - Irrefleksiv hvis vi ikke har noen $x \in A$ slik at xRx .
 - Symmetrisk hvis xRy medfører yRx for alle $x, y \in A$.
 - Antisymmetrisk hvis $xRy \wedge yRx$ medfører at $x = y$ for alle $x, y \in A$.
 - Transitiv hvis $xRy \wedge yRz$ medfører xRz for alle $x, y, z \in A$.

Oppsummering

- Etter å ha sett på endel eksempler på relasjoner, og deres tilknytning til egenskaper som *refleksiv*, *irrefleksiv*, *symmetrisk*, *antisymmetrisk* og *transitiv*, begynte vi å se på ekvivalensrelasjoner.
- En *ekvivalensrelasjon* er normalt en relasjon som uttrykker at objektene har visse felles egenskaper.
- Den formelle definisjonen var:

Ekvivalensrelasjoner

Definisjon.

La R være en relasjon på en mengde A .

Vi kaller R en ekvivalensrelasjon om R er refleksiv, symmetrisk og transitiv.

Merk.

- Vi rakk å se på del a) og del b) i følgende eksempel:

Ekvivalensrelasjoner

Eksempel.

- a) La A være mengden av sammensatte utsagn i utsagnsvariablene p_1, p_2, p_3 . Vi lar $\phi R \psi$ hvis $\phi \Leftrightarrow \psi$, det vil si hvis $\phi \leftrightarrow \psi$ er en tautologi.

Da er R en ekvivalensrelasjon.

- b) En *vektor* er et par (x, y) av punkter i planet eller rommet.

x kalles *roten* til vektoren og y kalles *spissen* til vektoren.

Det er vanlig å si at to vektorer er like hvis de har samme lengde og retning.

Det uttrykkes vi ved

$$(x, y)R(u, v) \Leftrightarrow y - x = v - u.$$

Dette er en ekvivalensrelasjon.

Ekvivalensrelasjoner

Eksempel (Fortsatt).

- c) Hvis vi arbeider med en stor programpakke, og ønsker å gjøre den mer effektiv, kan vi vinne en del på å erstatte en subrutine med en raskere en.

Hvis det for eksempel ofte inngår at data må ordnes på en bestemt måte, spiller det ikke så stor rolle for utfallet hvordan vi organiserer en slik ordning, men det kan bety mye for effektiviteten.

To mulige subrutiner er *ekvivalente* hvis de alltid omgjør de samme input-dataene til de samme output-dataene.

Dette er et eksempel på en ekvivalensrelasjon.

Ekvivalensrelasjoner

Merk.

- I forbindelse med eksempel c) kan vi bemerke:

Erstatter vi et underprogram i et program med et ekvivalent underprogram, blir det omskrevne programmet ekvivalent med det opprinnelige.

- Dette er et eksempel på en nyttig egenskap, men hvor vi trenger både ekvivalensrelasjoner og induksjonsbevis for å gi et skikkelig bevis.

Ekvivalensrelasjoner

En viktig egenskap ved en ekvivalensrelasjon R på en mengde A er at R deler A opp i disjunkte mengder av parvis ekvivalente elementer i A .

(Disjunkt betyr at snittet er tomt.)

Disse mengdene kaller vi ekvivalensklasser

Vi skal vise denne egenskaper ved ekvivalensrelasjoner helt generelt, men la oss se på noen eksempler først.

Ekvivalensrelasjoner

Eksempel.

- La $A = \{0, 1, 2, 3, 4, 5\}$
- La $R = \{(0, 0), (0, 1), (1, 1), (2, 2), (3, 3), (3, 4), (3, 5), (4, 3), (4, 4), (4, 5), (5, 3), (5, 4), (5, 5)\}$
- I utgangspunktet er det ikke så lett å se hvilke egenskaper R har, men beskriver vi R på matriseform blir bildet klarere:

Ekvivalensrelasjoner

Eksempel (Fortsatt).

$$\begin{bmatrix} T & T & F & F & F & F \\ T & T & F & F & F & F \\ F & F & T & F & F & F \\ F & F & F & T & T & T \\ F & F & F & T & T & T \\ F & F & F & T & T & T \end{bmatrix}$$

Vi ser at aRb hvis og bare hvis a og b tilhører den samme av delmengdene $\{0, 1\}$, $\{2\}$ og $\{3, 4, 5\}$.

Ekvivalensrelasjoner

Eksempel.

- Vi definerer relasjonen R på \mathbb{R}^2 ved $(x, y)R(u, v)$ om $x + y = u + v$.
- R er en ekvivalensrelasjon.
- Hvis $x + y = k$ vil $(x, y)R(u, v)$ om $u + v = k$.
- Denne relasjonen deler planet opp i uendelig mange disjunkte deler, hvor delene er alle linjene med stigningstall -1

Ekvivalensrelasjoner

Definisjon.

La R være en ekvivalensrelasjon på en mengde A og la $a \in A$.

Vi lar ekvivalensklassen til a være

$$E(a) = \{b \in A \mid aRb\}.$$

Ekvivalensrelasjoner

Teorem.

La R være en ekvivalensrelasjon på mengden A , $E(a)$ ekvivalensklassen til $a \in A$.

- a) For alle $a \in A$ vil $a \in E(a)$.
- b) Hvis aRb vil $E(a) = E(b)$.
- c) Hvis $\neg(aRb)$ vil $E(a) \cap E(b) = \emptyset$

Ekvivalensrelasjoner

Dette teoremet sier nettopp at hvis R er en ekvivalensrelasjon på A , så kan vi dele A opp i disjunkte klasser av parvis ekvivalente elementer.

Bevis.

Siden aRa vil $a \in E(a)$. Dette viser a).

La aRb . Skal vise at $E(a) = E(b)$.

Siden vi da også har at bRa er det nok å vise at $E(b) \subseteq E(a)$ for å vise b).

Så la $c \in E(b)$.

Da vil $aRb \wedge bRc$ så aRc fordi R er transitiv.

Det betyr at $c \in E(a)$.

Siden $c \in E(b)$ var valgt vilkårlig, kan vi slutte at $E(b) \subseteq E(a)$.

Ekvivalensrelasjoner

Bevis (Fortsatt).

Til sist, anta at $c \in E(a) \cap E(b)$.

Da vil $aRc \wedge bRc$.

Ved symmetri og transitivitet for R følger det at aRb .

Snur vi dette argumentet på hodet, får vi at $\neg(aRb) \Rightarrow E(a) \cap E(b) = \emptyset$.

Dette viser c), og teoremet er bevist.

Ekvivalensrelasjoner

Oppgave.

La A være en mengde og la R og S være ekvivalensrelasjoner på A .

La $T = S \cap R$

- Forklar hvorfor vi har at aTb hvis og bare hvis $aRb \wedge aSb$ for alle a og b i A .
- Vis at T er en ekvivalensrelasjon.
- Finn et eksempel hvor $R \cup S$ ikke er en ekvivalensrelasjon.

Ordninger

- En annen type relasjoner som forekommer ofte er ordninger.
- Vi har en ordning når vi formaliserer "mindre eller lik", "er svakere enn" og tilsvarende forhold.
- Vi tar definisjonen først, og illustrerer den med noen eksempler etterpå.
- Vi følger boka og definerer:

Ordninger

Definisjon.

La A være en mengde med en relasjon R .

Vi kaller R en partiell ordning hvis

1. R er refleksiv
2. R er transitiv
3. R er antisymmetrisk.

Ordninger

Eksempel.

- a) La \mathcal{E} være en universell mengde, og \subseteq være inklusjonsordningen på potensmengden til \mathcal{E} .
1. \subseteq er *refleksiv* fordi $A \subseteq A$ for alle mengder A .
 2. \subseteq er *transitiv* fordi $A \subseteq B$ og $B \subseteq C$ alltid vil medføre at $A \subseteq C$
 3. \subseteq er *antisymmetrisk* fordi $A = B$ når $A \subseteq B$ og $B \subseteq A$.

Dette viser at \subseteq er en partiell ordning.

Ordninger

Eksempel (Fortsatt).

- b) La \leq være den vanlige "mindre-eller-lik" relasjonen på \mathbb{J} .
 \leq er opplagt både refleksiv, transitiv og antisymmetrisk, så \leq er en partiell ordning.
 $<$ er derimot *IKKE* en partiell ordning, fordi $<$ ikke er *refleksiv*.
- c) Hvis vi ordner studentene i MAT1030 etter oppnådd karakter, med A øverst, F langt nede, men "ikke møtt" og "trukket seg før eksamen" enda lenger nede, har vi ikke en partiell ordning.
Siden det er mer enn 8 studenter, må minst to stykker lide samme MAT1030-skjebne.
Disse vil stå likt, men være forskjellige personer.
Det betyr at denne relasjonen ikke er *antisymmetrisk*

Ordninger

- Det er en viktig forskjell mellom ordningen av potensmengden til \mathcal{E} og ordningen av \mathbb{J} .
- Hvis vi har to tall a og b vil en av tre holde:
 1. $a = b$
 2. $a < b$
 3. $b < a$
- For inklusjon har vi en fjerde mulighet, nemlig at ingen av A eller B er inneholdt i den andre.
- Vi fanger opp denne forskjellen i en definisjon:

Ordninger

Definisjon.

La R være en partiell ordning på en mengde A .
 R kalles total dersom vi for alle a og b i A har at

$$aRb \vee bRa.$$

Merk.

- Det er for totale ordninger at vi har gode sorteringsalgoritmer.
- Utviklingen av slike algoritmer overlater vi til foreleserne i programmering, de er flinkere til å finne effektive algoritmer.

En utfordring

- Dette er en oppgave for de som har lyst til å se hvor gode de er til å resonere rundt de generelle definisjonene vi har gitt.
- Vi forventer ikke at studentene skal kunne løse slike oppgaver til eksamen.

Oppgave.

La R være en relasjon på en mengde A .
Vi kaller R en preordning hvis R er transitiv og refleksiv.
Definer relasjonen S på A ved aSb når $aRb \wedge bRa$.

- a) Vis at S er en ekvivalensrelasjon på A .

Der er vanlig å skrive A/S for mengden av ekvivalensklasser $E(a)$ til ekvivalensrelasjonen S .

En utfordring

Oppgave (Fortsatt).

Vi definerer en relasjon \hat{R} på A/S ved

$$E(a)\hat{R}E(b)$$

om aRb

- b) Vis at det ikke spiller noen rolle hvilke elementer i ekvivalensklassene $E(a)$ og $E(b)$ vi bruker.
- c) Vis at \hat{R} er en partiell ordning på mengden av ekvivalensklasser.

OVER TIL KAPITTEL 6

Funksjoner

- Fra skolematematikken kjenner man funksjoner definert ved uttrykk som
 - $f(x) = x^2 + 3x + 4$
 - $g(x) = \sin x$
 - $h(x) = e^{2\ln(x)+27}$
- Vi har også støtt på tallteoretiske funksjoner som
 - Heltallsverdien til $\frac{n}{m}$
 - Primtall nr. n .
 - Største felles mål av n og m .

Funksjoner

Felles for alle disse eksemplene er at det er en sammenheng mellom argumentet eller argumentene til funksjonen, og verdien til funksjonen.

Funksjoner

Det er vanlig å fremstille en funksjon som en slags svart boks hvor

- Vi fører boksen med funksjonsargumentene.
- Boksen bearbeider dataene.
- Boksen gir oss funksjonsverdien.

Som intuitivt bilde er dette en grei beskrivelse.

Som presis matematisk definisjon holder det ikke helt.

Vi skal holde oss til presisjonsnivået i boka.

Hovedpoenget er at vi skal tolke ordet *regel* på neste side så liberalt som mulig.

Funksjoner

Definisjon.

La X og Y være to mengder.

En funksjon $f : X \rightarrow Y$ er en regel som for hver $x \in X$ gir oss en, og bare en, $y = f(x)$ i Y .

Merk.

- Som sagt skal ordet *regel* tolkes så liberalt som mulig, men, som vi skal se, med en smule forsiktighet.
- Det er ikke noe krav om at det skal foreligge noen regler som mennesker eller maskiner kan følge.
- Det eneste kravet er at $f(x)$ skal finnes i Y , og at uttrykket $f(x)$ ikke kan være flertydig.

Funksjoner

Eksempel (Funksjoner).

- Vi skal se på noe av det vi har snakket om før, og se hvorvidt funksjoner er involvert.
- Vi skal legge vekt på andre eksempler enn de dere kjenner fra skolematematikken.
- La \mathcal{E} være en universell mengde, og la A være en delmengde av \mathcal{E}
- Vi definerte $\bar{A} = \{x \in \mathcal{E} : x \notin A\}$.
- Hver A har en og bare en komplement-mengde, så $f(A) = \bar{A}$ er en funksjon.
- Her er X lik Y lik potensmengden til \mathcal{E} .

Funksjoner

Eksempel (Fortsatt).

- På samme måte kan vi oppfatte \cap og \cup som funksjoner.
- Hvis Y er potensmengden til \mathcal{E} og $X = Y^2$, vil \cap og \cup være funksjoner fra X til Y .
- Vi har sett på binærrepresentasjonen til hele tall.
- I hvilken forstand er

$\text{REP}(a) = \text{binærrepresentasjonen til } a$

en funksjon?

Funksjoner

Eksempel (Fortsatt).

- Vi vet at ikke alle tall har en binær representasjon.
- Det betyr at REP ikke oppfyller kravet

$$\forall x \in X \exists y \in Y (\text{REP}(x) = y)$$

hvis vi lar $X = \mathbb{J}$.

- Lar vi X være mengden av tall med en binær representasjon, blir REP en funksjon.

Funksjoner

Eksempel (Fortsatt).

- Quicksort er en algoritme som sorterer elementene i en liste etter størrelse.
- Quicksort gir mening hver gang listen er hentet fra en totalt ordnet mengde.
- Da beregner Quicksort funksjonen som tar en vilkårlig liste som argument og gir den sorterte listen som verdi.
- Vi kan finne en annen algoritme Slowsort for sortering av elementene i en liste, og den vil definere den samme funksjonen, men være en annen algoritme.
- Det er forbindelsen mellom argument og verdi som bestemmer hvilken funksjon vi har, ikke hvordan vi kommer fra argument til verdi.

Funksjoner

Definisjon.

- a) Hvis $f : X \rightarrow Y$ er en funksjon kaller vi
 - X for definisjonsområdet til f .
 - Y for verdiområdet til f .
- b) Bildet eller bildemengden til f er

$$\{f(x) : x \in X\}$$

- c) Vi kan skrive $f[X]$ for bildet til f .

Funksjoner

Eksempel (Funksjoner).

- Definer $f(x) = e^x$ som en funksjon fra \mathbb{R} til \mathbb{R} .

Da vil:

- *Definisjonsområdet* til f være hele \mathbb{R} .
- *Verdiområdet* til f være hele \mathbb{R} .
- *Bildet* til f være mengden av positive reelle tall.

Funksjoner

Eksempel (Fortsatt).

- La $A \subseteq \mathcal{E}$ være en mengde, og definer

$$f_A(B) = A \cap B$$

som en funksjon fra potensmengden X til \mathcal{E} til seg selv.

- Da er X både definisjonsområdet og verdiområdet til f_A , mens bildemengden vil være potensmengden til A .

Funksjoner

Eksempel (Fortsatt).

- Innledningsvis i disse forelesningene konstruerte vi en pseudokode for å beregne $|n - m|$ fra n og m .
- I dette tilfellet var definisjonsområdet mengden av par av ikke-negative hele tall, verdiområdet og bildemengden lik mengden av ikke-negative hele tall.
- Da vi satte opp sannhetsverditabeller for sammensatte utsagn i utsagnsvariablene p , q og r , konstruerte vi i virkeligheten funksjoner fra mengden X av tripler fra $\{T, F\}$ til $Y = \{T, F\}$.

To sammensatte utsagn er logisk ekvivalente når disse funksjonene er like.

Funksjoner

Merk.

- Læreboka bruker *domain* for definisjonsområde og *codomain* for verdiområde.
- Det er ikke uvanlig å bruke ordene *domene* og *kodomene* på norsk.
- Vi skal holde oss til betegnelsene *definisjonsområde* og *verdiområde* i disse forelesningene.

Funksjoner

- Funksjoner har en ting felles med relasjoner:
Noen av dem har spesielle egenskaper som er verd en egen betegnelse.
- Vi skal først se på injektive funksjoner.
- Andre betegnelser er 1-1-funksjoner og enentydige funksjoner.

Funksjoner

Eksempel (Injektive funksjoner).

- La w være et 32-bits binært tall, og la $f(w)$ være det heltallet som har w som 4-bytes representasjon.
- Hvis $v \neq w$ vil $f(v) \neq f(w)$ siden vi ikke bruker to forskjellige binære representasjoner av det samme tallet.
- Hvis vi oppfatter f som en *input-output*-forbindelse, ser vi at det er én input pr. output.
- Dette er et eksempel på en en-til-en-funksjon, eller *injektiv* funksjon.

Funksjoner

Eksempel (Fortsatt).

- En telefonselger vil ha datastøttet oppringing til kunder.
- For ikke å irritere kundene unødige, bør ikke telefonselgeren kontakte samme kunde to ganger.
- Hvis $R(n)$ er kunde nummer n selgeren kontakter, betyr dette kravet at $R(n) \neq R(m)$ når $n \neq m$.
- Programmet selgeren støtter seg på må være slik at oppringningsfunksjonen R blir injektiv, eller enentydig.

Funksjoner

Den formelle definisjonen vil være:

Definisjon.

La $f : X \rightarrow Y$ være en funksjon.

f kalles injektiv hvis vi for alle x og y i X har at

$$x \neq y \Rightarrow f(x) \neq f(y).$$

Funksjoner

Merk.

- Vi har brukt den kontrapositive versjonen i definisjonen.
En ekvivalent formulering vil være

$$f(x) = f(y) \Rightarrow x = y.$$

Funksjoner

Eksempel (Injektive funksjoner).

- La $f(x) = x^2$ være en funksjon fra \mathbb{R} til \mathbb{R} .
Da er ikke f injektiv fordi $1 \neq -1$ mens $f(1) = f(-1)$.
- Hvis vi begrenser definisjonsområdet til de ikke-negative tallene $\mathbb{R}_{\geq 0}$ blir funksjonen injektiv.
- Funksjonen som ordner en sekvens av ord alfabetisk er ikke injektiv, fordi ordsekvensene "Per,Pål,Espen" og "Espen,Pål, Per" er forskjellige, men de blir like når vi ordner dem alfabetisk.

Forelesning 13

Funksjoner

Dag Normann - 25. februar 2008

Opphenting

- Forrige forelesning fortsatte vi innføringen av ekvivalensrelasjoner.
- Vi definerte hva vi mener med partielle ordninger og med totale ordninger.
- Deretter snakket vi generelt om funksjoner og spesielt om injektive funksjoner.
- Det ble mange nye begreper, og før vi fortsetter, skal vi repetere, ved hjelp av eksempler, noen av disse begrepene.
- Vi starter med å se på en funksjon og et par avledede relasjoner.

Opphenting

Eksempel.

- La $A = \{1, \dots, 100\}$
- La $f(a)$ være tverrsummen til $a \in A$ når vi antar at vi bruker vanlig titallsystem.
- f er en funksjon, hvor A er definisjonsområdet.
- I dette tilfellet har vi ikke bestemt verdiområdet, men uten å tenke oss om vet vi at tverrsummen vil være et naturlig tall.
- Vi ser derfor på f som en funksjon

$$f : A \rightarrow \mathbb{N}$$

Opphenting

Eksempel (Fortsatt).

- Er f injektiv?
- Kravet var at for alle a og b i A , så skal $f(a) \neq f(b)$ når $a \neq b$.
- Men $f(63) = f(72) = 9$, så f er ikke injektiv.
- Kan vi bestemme bildemengden til f ?
- Bildemengden vil være $B = \{1, 2, 3, 4, \dots, 18\}$.

Opphenting

Eksempel (Fortsatt).

- Vi definerer nå to relasjoner på A ved hjelp av f :
- La aRb hvis $f(a) = f(b)$.
- La aSb hvis $f(a) < f(b) \vee (aRb \wedge a \leq b)$
 - Er R eller S refleksiv?
 - Er R eller S irrefleksiv?
 - Er R eller S symmetrisk?
 - Er R eller S antisymmetrisk?
 - Er R eller S transitiv?

Opphenting

Eksempel (Fortsatt).

- La oss se på R først.
 - Vi ser at R er *refleksiv* fordi $f(a) = f(a)$ for alle $a \in A$.
 - Vi ser at R er *symmetrisk* fordi $f(b) = f(a)$ når $f(a) = f(b)$.
 - Vi ser at R er *transitiv* fordi $f(a) = f(c)$ hvis vi har at $f(a) = f(b)$ og at $f(b) = f(c)$.
 - Siden $A \neq \emptyset$ og R er refleksiv, er ikke R samtidig *irrefleksiv*.
 - Siden f ikke er injektiv og R er symmetrisk, kan ikke R være *antisymmetrisk*.
- Det betyr at R er en ekvivalensrelasjon.
- Vi har ikke brukt noen spesielle egenskaper ved A eller f , så relasjoner konstruert på denne måten vil alltid være ekvivalensrelasjoner.

Opphenting

Eksempel (Fortsatt).

- Når vi nå vet at R er en ekvivalensrelasjon, kan vi prøve å finne ekvivalensklassene.
- Vi vil ha en ekvivalensklasse for hver tverrsum:
 1. $\{1, 10, 100\}$
 2. $\{2, 11, 20\}$
 3. $\{3, 12, 21, 30\}$
 4. $\{4, 13, 22, 31, 40\}$
 5. $\{5, 14, 23, 32, 41, 50\}$osv.

Opphenting

Eksempel (Fortsatt).

- La oss så se på egenskapene til S .
- $aSb \Leftrightarrow f(a) < f(b) \vee (aRb \wedge a \leq b)$
 - S er *refleksiv* fordi $aRa \wedge a \leq a$ for alle a
 - La aSb og bSc .
 - * Hvis $f(a) < f(b)$ eller $f(b) < f(c)$ vil $f(a) < f(c)$, og aSc .
 - * Hvis $f(a) = f(b) = f(c)$ har vi at $a \leq b \leq c$, så da har vi også at aSc .
 - Vi ser at S er transitiv.
 - Anta at aSb og bSa
 - * Da må $f(a) \leq f(b)$ og $f(b) \leq f(a)$, så $f(a) = f(b)$, det vil si at aRb .
 - * Da er $a \leq b$ og $b \leq a$ så $a = b$
 - Det følger at S er *antisymmetrisk*.
- Konklusjonen er at S er en partiell ordning.
- Oppgave: Vis at S er en total ordning, og skriv ned de 10 S -første tallene.

Opphenting

Eksempel (Injektive funksjoner).

- Vi skal se på tre funksjoner, og spørre om de er injektive:
 1. A er alle ord w godkjent som norske ord, og $f(w)$ er ordet w skrevet baklengs.
 2. B er mengden av uendelige desimalutviklinger α og $g(\alpha)$ er det tilsvarende reelle tallet.
 3. C er mengden av positive reelle tall som har en eksakt 32-bits representasjon r , og hvis r representerer tallet x lar vi $h(r)$ være tallet som representerer \sqrt{x} .
- f vil være injektiv, for hvis vi speiler to forskjellige ord, vil speilbildene bli forskjellige.

Opphenting

Eksempel (Fortsatt).

- g er ikke injektiv fordi noen reelle tall kan ha to forskjellige desimalutviklinger, eksempelvis

$$0,9999\dots = 1,0000\dots$$

- h kan ikke være injektiv fordi hvis et tall ligger mellom 2^{-24} og 2^{24} , vil kvadratrotten ligge mellom 2^{-12} og 2^{12} .

Vi har altså langt færre binære tall til disposisjon for å representere \sqrt{x} enn x selv, så funksjonen kan ikke være injektiv.

- Her har vi egentlig brukt noe som kalles skuffeprinsippet, dueslagsprinsippet eller på engelsk the pigeon hole principle.

Vi er nå ferdige med opphentingene, og fortsetter der vi slapp på onsdag.

Funksjoner

Den neste gruppen av funksjoner vi skal se på er de surjektive funksjonene:

Definisjon.

La $f : X \rightarrow Y$ være en funksjon.

f kalles surjektiv hvis bildemengden til f er hele Y .

På engelsk brukes ofte betegnelsen onto og på norsk kan vi si at f går fra X på Y .

Funksjoner

Eksempel (Surjektive funksjoner).

- La $f : \mathbb{R} \rightarrow \mathbb{R}_{\geq 0}$ være gitt ved

$$f(x) = x^2.$$

Da er f surjektiv.

- $f_A(B) = A \cap B$ er surjektiv som en funksjon fra potensmengden til \mathcal{E} til potensmengden til A .
- La $\text{PRIM} : \mathbb{N} \rightarrow \mathbb{N}$ være definert ved at $\text{PRIM}(n)$ er primtall nr. n .

Da er ikke PRIM en surjektiv funksjon, fordi verdimengden er hele \mathbb{N} , mens bildemengden er mengden av primtall.

Funksjoner

Eksempel (Fortsatt).

- Enhver funksjon vil være surjektiv hvis vi setter verdiområdet lik bildemengden.
- Det er ofte i de tilfellene hvor det er uklart hva bildemengden er at det kan være relevant å spørre seg om en funksjon er surjektiv eller ikke.
- Funksjonen på neste side er surjektiv, men det er det svært krevende å vise:

Funksjoner

Eksempel (Surjektive funksjoner).

Vi definerer $f : \mathbb{N} \rightarrow \mathbb{N}$ ved

- $f(n) = m$ hvis $2 < m \leq n$, det finnes a, b og c i \mathbb{N} slik at

$$a^m + b^m = c^m$$

og slik at m er det minste tallet med denne egenskapen.

- $f(n) = n$ hvis det ikke finnes noen slik $m \leq n$.

Funksjoner

Merk.

- I MAT1030 vil injektive funksjoner spille en større rolle enn surjektive funksjoner.
- Hvis vi konstruerer digitale representasjoner for elementene i en mengde A , konstruerer vi samtidig en funksjon F fra representantene for elementene i A til A selv.
- Vi vil normalt ønske å vite om vi får representert alle elementene i A (håpløst hvis ikke A er endelig).
- Dette er det samme som å spørre om F er surjektiv.

Funksjoner

Eksempel (Sammensatte funksjoner).

- La oss holde oss til lekedatamaskinen fra innledningen og la a være et helt tall slik at

$$-127 \leq a \leq 127.$$

- Hvis vi skal beregne $f(a) = -a$ på datamaskinen trenger vi å
 1. Finne den digitale representasjonen av a .
 2. Beregne den digitale representasjonen av $-a$ fra den digitale representasjonen til a
 3. Finne $-a$ ut fra den digitale representasjonen til $-a$.
- Vi ser at det er tre funksjoner involvert her, og vi har bruk for sammensetningen av dem.

Funksjoner

Eksempel (Fortsatt).

- I skolematematikken, og i noen kurs i matematisk analyse, ser vi på sammensetninger av funksjoner definert på \mathbb{R} eller delmengder av \mathbb{R} .
- Kjernerregelen for derivasjon forteller oss hvordan vi kan derivere slike sammensetninger.

Funksjoner

Eksempel (Sammensatte funksjoner).

- La oss se på følgende pseudokode.

1. *Input* x [x naturlig tall]

2. $y \leftarrow 1$

3. **While** $x > 0$ **do**

 3.1 $y \leftarrow 2y$

 3.2 $x \leftarrow x - 1$

4. $z \leftarrow 1$

5. **While** $y > 0$ **do**

 5.1 $z \leftarrow 3z + 1$

 5.2 $y \leftarrow y - 1$

6. *Output* z

Funksjoner

Eksempel (Fortsatt).

- Denne pseudokoden deler seg naturlig i to deler.
- Instruksjonene 1. - 3. beregner $y = f(x) = 2^x$.
- Instruksjonene 4. - 6. beregner z som en funksjon $z = g(y)$, hvor vi ikke har noen opplagt formel.
- Tilsammen vil pseudokoden definere en sammensatt funksjon $z = g(f(x))$.

Funksjoner

Definisjon.

La $f : X \rightarrow Y$ og $g : Y \rightarrow Z$ være to funksjoner.

Vi definerer sammensetningen $h = g \circ f$ som funksjonen

$$h : X \rightarrow Z$$

vi får ved først å bruke f på argumentet x og så g på mellomverdien $f(x)$.

Vi skriver også

$$h(x) = g(f(x)).$$

Funksjoner

Eksempel.

- La $f : \mathbb{N} \rightarrow \mathbb{N}$ være definert ved at $f(n)$ er primtall nummer n og la $g : \mathbb{N} \rightarrow \mathbb{N}$ være definert ved at $g(m) = m^2$

Da er $(g \circ f)(4) = g(f(4)) = g(7) = 49$.

$(f \circ g)(4) = f(g(4)) = f(16) = 53$.

- Vi ser altså at selv om sammensetningen av funksjonene gir mening for begge rekkefølgene, kan det spille en rolle i hvilken rekkefølge vi setter dem sammen.

Funksjoner

Eksempel (Fortsatt).

- La f sende binærformen til et naturlig tall n over til desimalformen og la g gi oss tverrsummen av desimalformen til et tall.

Da gir det ingen mening å snakke om $f \circ g$ fordi definisjonsområdet til f er mengden av binære representasjoner av naturlige tall og verdiområdet til g er en mengde av naturlige tall.

I MAT1030 er det viktig å skille mellom tall og de forskjellige representasjonene av tallene.

$g \circ f$ gir mening. Definisjonsområdet vil være mengden av binære representasjoner, "mellomområdet" vil være mengden av desimaltallsrepresentasjoner og verdiområdet vil være naturlige tall.

$(g \circ f)(100110) = g(38) = 11$.

Funksjoner

Teorem.

La $f : X \rightarrow Y$ og $g : Y \rightarrow Z$ være to funksjoner.

La $h = g \circ f$ være sammensetningen av f og g .

- Hvis både f og g er injektive, er h injektiv.
- Hvis både f og g er surjektive, er h surjektiv.

Funksjoner

Bevis.

- Anta at f og g er injektive.

$$x \neq y \Rightarrow f(x) \neq f(y) \Rightarrow g(f(x)) \neq g(f(y)).$$

Siden \Rightarrow er *transitiv*, $h(x) = g(f(x))$ og $h(y) = g(f(y))$ følger det at h er injektiv når f og g er det.

- Anta at f og g er surjektive, og la $z \in Z$ være vilkårlig.

Siden g er surjektiv, finnes $y \in Y$ slik at $g(y) = z$.

Siden f er surjektiv, finnes $x \in X$ slik at $f(x) = y$.

Da er $z = g(f(x)) = h(x)$.

Funksjoner

Eksempel (Inverse funksjoner).

- Med fare for å overfokusere på et tema skal vi nok en gang se på digital representasjon av tall.
- La X være mengden av reelle tall som har en digital representasjon med enkel presisjon.
- La Y være mengden av 32 bits representasjoner av reelle tall.
- La $F : X \rightarrow Y$ være funksjonen som til et tall x gir oss den digitale representasjonen av x .
- La $G : Y \rightarrow X$ være funksjonen som til en digital representasjon y av et tall gir oss tallet.
- Da er $G(F(x)) = x$ for alle $x \in X$ og $F(G(y)) = y$ for alle $y \in Y$.
- Vi sier at G er den inverse av F .

Funksjoner

Eksempel (Inverse funksjoner).

- Fra skolematematikken og begynneremnene i matematikk har vi flere eksempler på par av funksjoner som "opphever" hverandre:

1. $f : \mathbb{R} \rightarrow \mathbb{R}_{>0}$ definert ved $f(x) = e^x$ og $g : \mathbb{R}_{>0} \rightarrow \mathbb{R}$ definert ved $g(y) = \ln(y)$.
 2. Tangensfunksjonen $f : \langle -\frac{\pi}{2}, \frac{\pi}{2} \rangle \rightarrow \mathbb{R}$ og Arcustangens-funksjonen $g : \mathbb{R} \rightarrow \langle -\frac{\pi}{2}, \frac{\pi}{2} \rangle$.
 3. $f(x) = 2x$ og $g(y) = \frac{y}{2}$
- Vi har utallige eksempler på par av funksjoner som representerer "omvendte regningsmåter av hverandre".

Funksjoner

Eksempel (Inverse funksjoner).

- La $f : \mathbb{J} \rightarrow \mathbb{J}$ være definert ved at $f(a)$ er heltallsverdier til $\frac{a}{2}$, det vil si det største hele tallet b slik at $b \leq \frac{a}{2}$.
- Kan vi finne en omvendning av f ?
- La oss regne på noen verdier, og la oss se hva som skjer:
- $\dots, f(-2) = f(-1) = -1, f(0) = f(1) = 0, f(2) = f(3) = 1, \dots$
- Hvis vi skal lage en omvendt funksjon g har vi to valg for $g(1)$, nemlig $g(1) = 2$ og $g(1) = 3$.
- Velger vi $g(1) = 2$, vil $g(f(3)) \neq 3$, og velger vi $g(1) = 3$ får vi $g(f(2)) \neq 2$.
- Vi ser at siden f ikke er injektiv, får vi et problem.

Funksjoner

Eksempel (Inverse funksjoner).

- La $P : \mathbb{N} \rightarrow \mathbb{N}$ være definert ved at $P(n)$ er primtall nummer n i den voksende opplistingen

$$2, 3, 5, 7, 11, 13, 17, 19, \dots$$
- P er injektiv, men P har ingen omvendt funksjon Q .
- Problemet er at vi ikke kan definere eksempelvis $Q(15)$ siden 15 ikke er et primtall og derfor ikke har noe nummer.
- Vi ser at problemet ligger i at O ikke er surjektiv.

Funksjoner

Definisjon.

La $f : X \rightarrow Y$ være en funksjon.

$g : Y \rightarrow X$ kalles en invers til f , eller en omvendt funksjon av f , hvis

- $f(g(x)) = x$ for alle $x \in X$.
- $g(f(y)) = y$ for alle $y \in Y$.

Funksjoner

Teorem.

a) La $f : X \rightarrow Y$ være en funksjon.

Da har f en invers funksjon $g : Y \rightarrow X$ hvis og bare hvis f er både injektiv og surjektiv.

b) En funksjon f kan ikke ha mere enn en invers.

Definisjon.

Hvis $f : X \rightarrow Y$ har en invers, skriver vi f^{-1} for den inverse.

Funksjoner

Bevis.

- Anta først at $f : X \rightarrow Y$ er både injektiv og surjektiv.

La $y \in Y$.

Siden f er surjektiv, finnes det $x \in X$ slik at $f(x) = y$, og siden f er injektiv finnes det bare en slik x .

Da lar vi $g(y) = x$, og g vil være en invers.

Funksjoner

Bevis (Fortsatt).

- Anta så at f har en invers g .

$$- x \neq y \Rightarrow g(f(x)) \neq g(f(y)) \Rightarrow f(x) \neq f(y).$$

Derfor er f injektiv.

- La $y \in Y$ og la $x = g(y)$.

Da er $y = f(x)$.

Det følger at f er surjektiv.

- Definisjonen under første punkt er den eneste måten å finne en invers på, så det kan ikke finnes flere.

Funksjoner

Oppgave.

- a) Vis at hvis $f : X \rightarrow Y$ og $g : Y \rightarrow Z$ begge har inverse f^{-1} og g^{-1} , så vil sammensetningen

$$h = g \circ f$$

også ha en invers.

- b) Under antagelsene i a), vis at $h^{-1} = f^{-1} \circ g^{-1}$.

Funksjoner

- Vi kan bruke kvantorer til å gi presise definisjoner av *injektiv*, *surjektiv* og *sammensetning*:
- $f : X \rightarrow Y$ er injektiv hvis $\forall x \in X \forall y \in X (f(x) = f(y) \rightarrow x = y)$.
- $f : X \rightarrow Y$ er surjektiv hvis $\forall y \in Y \exists x \in X (y = f(x))$
- Hvis $f : X \rightarrow Y$, $g : Y \rightarrow Z$ og $h = g \circ f$ vil

$$\forall x \in X \forall z \in Z (h(x) = z \leftrightarrow \exists y \in Y (y = f(x) \wedge z = g(y))).$$

De som ønsker å forstå hvordan kvantorer brukes, bør overbevise seg selv om at dette er riktig.

Funksjoner og programmering

- De fleste programmeringsspråk er utstyrt med predefinerte funksjoner.
- For noen av disse språkene kan man også definere sine egne funksjoner.
- En lommeregner har eksempelvis knapper for algebraiske operasjoner, trigonometriske funksjoner, logaritme- og eksponensialfunksjoner m.m.
- Noen lommeregnere tillater også at vi definerer våre egne funksjoner ved hjelp av sammensatte uttrykk, og tildels enkle programmer.
- Matematisk sett opererer disse funksjonene på representasjoner av tall og andre størrelser i lommeregneren eller på datamaskinen.
- Dette diskuteres i tilstrekkelig detalj i læreboka.

Funksjoner og programmering

- Noen hjelpemidler kan gå under betegnelsen "funksjoner" i en programmeringssammenheng, men er ikke funksjoner i vanlig matematisk forstand.
- De viktigste er de som genererer et tilfeldig element i en mengde.
- Ber vi om en kabal, eller et minesveiperspill, får vi et nytt spill hver gang.
- Skal vi teste om et stort tall n er et primtall utfører vi en algebraisk test på tilfeldig valgte tall a_1, \dots, a_k mindre enn n .

For hver test vil svaret *NEI* fortelle oss at n ikke er et primtall, mens svaret *JA* forteller oss at sannsynligheten er $\frac{3}{4}$ for at n er et primtall.

- Denne primtallstesten kan gi forskjellige svar på om n er et primtall, og er derfor ikke en funksjon i matematisk forstand.
- Ved å la antall vilkårlige deltester være stor, eksempelvis 100, er usikkerheten omkring svaret i størrelsesorden 2^{-200} , så for alle praktiske formål er primtallstesten en funksjon.

Forelesning 14

Rekursjon og induksjon

Dag Normann - 27. februar 2008

Oppsummering

- Mandag repeterte vi en del om relasjoner, da spesielt om ekvivalensrelasjoner og partielle ordninger.
- Vi snakket videre om funksjoner.
- Det er noen grunnleggende begreper i tilknytning til kapitlet om funksjoner man må kjenne til for å kunne gå eksamensdagen i møte med ro i sinnet.
- Det er
 - Injektive funksjoner, også kalt 1-1-funksjoner eller enetydige funksjoner.
 - Surjektive funksjoner (onto).
 - Sammensetning av funksjoner.
 - Omvendte eller inverse funksjoner.

Oppsummering

Det er også viktig å holde orden på hva som menes med:

- *Definisjonsområdet* til en funksjon.
- *Verdiområdet* til en funksjon.
- *Bildemengden* til en funksjon.

I tillegg bør man kunne vite når

- man kan finne en invers til en funksjon
- man kan sette sammen to funksjoner.

Dette avslutter den abstrakte innføringen i funksjoner.

Før vi går over til neste kapittel skal vi imidlertid se litt på hva det vil si at en funksjon er beregnbar.

Beregnbare funksjoner

- IT dreier seg mye om hvordan man løser oppgaver ved hjelp av elektroniske hjelpemidler, fortrinnsvis datamaskiner.
- All IT-aktivitet på maskin-nivå styres av programmer, uansett om vi ser dem eller ikke.
- Hvis man skal kunne forstå informasjonsteknologiens begrensninger, må vi derfor forstå grensene for hva det er mulig å skrive programmer for.
- Alle programmer beskriver egentlig funksjoner, selv om noen argumenter (som maskintid, maskinarkitektur o.a.) ikke er synlig.

- Det er derfor av interesse å studere de funksjonene som lar seg uttrykke ved hjelp av programmer.

Beregnbare funksjoner

- Hvis vi begrenser oss til funksjoner fra \mathbb{N}_0 til \mathbb{N}_0 har vi gode matematiske karakteriseringer av de beregnbare funksjonene, det vil si de som kan programmeres i et eller annet programmeringsspråk. ($\mathbb{N}_0 = \mathbb{N} \cup \{0\}$)
- Det viser seg at alle programmerbare funksjoner fra \mathbb{N}_0 til \mathbb{N}_0 kan formuleres som en av våre pseudokoder, hvor vi bare bruker navn på tallene 0 og 1, addisjon og multiplikasjon og Booleske tester uttrykt ved hjelp av = og <.

Det er ikke uvanlig for logikere eller folk som arbeider med teoretisk databehandling å la de naturlige tallene starte med 0.

Vi skal være snille og holde oss til måten boka gjør det på.

Beregnbare funksjoner

Som en forberedelse til kapittel 7 om induksjon og rekursjon, skal vi se på to pseudokoder hvor vi har pålagt oss å begrense oss til addisjon, multiplikasjon og Booleske tester med = og < (men dermed får lov til å bruke \leq).

- I det første eksemplet skal vi beregne $f(x, y) = \max\{0, x - y\}$.
- I det andre eksemplet skal vi beregne $g(x, y) = x^y$.

Beregnbare funksjoner

Eksempel (Beregnbare funksjoner).

1. *Input* x [$x \in \mathbb{N}_0$]
2. *Input* y [$y \in \mathbb{N}_0$]
3. $z \leftarrow 0$
4. **While** $y < x$ **do**
 - 4.1 $y \leftarrow y + 1$
 - 4.2 $z \leftarrow z + 1$
5. *Output* z

- Vi har ikke snakket om induksjonsbevis ennå. Det vil være den naturlige metoden for å vise korrekthet av et slikt program.
- I dette tilfellet ser vi at hvis $x \leq y$ starter vi ikke løkka i det hele tatt, mens hvis $y < x$ "teller" vi y opp til x samtidig som vi øker verdien av z tilsvarende mye.

Beregnbare funksjoner

Eksempel (Beregnbare funksjoner).

1. *Input* x [$x \in \mathbb{N}_0$]
2. *Input* y [$y \in \mathbb{N}_0$]
3. $u \leftarrow 0$
4. $z \leftarrow 1$
5. **While** $u < y$ **do**
 - 5.1 $z \leftarrow z \cdot x$
 - 5.2 $u \leftarrow u + 1$
6. *Output* z

Dette resulterer i at vi multipliserer x med seg selv y ganger, altså at vi beregner x^y .

Beregnbare funksjoner

- I programmeringssammenheng er det ikke alltid så lett å vite når et gitt program med et gitt input faktisk gir oss et output i den mengden hvor vi vil ha det.
- I verste fall kan vi skrive programmer for funksjoner hvor det er umulig å bestemme hva definisjonsområdet er.
- Innenfor IT er det derfor naturlig også å studere partielle funksjoner fra en mengde X til en mengde Y .
- Dette vil være funksjoner hvor definisjonsområdet er en delmengde av X og hvor verdiområdet er Y .
- Tolkningen av et program som en funksjon fra et Cartesisk produkt av datatyper til en datatype vil vanligvis være som en partiell funksjon.

OVER TIL KAPITTEL 7

Innledning til rekursjon og induksjon

- Vi skal nå starte på avsnittet om rekursive konstruksjoner og bevis ved induksjon.
- Dette er det første stedet hvor årets MAT1030 vil omfatte mer stoff enn det læreboka omfatter.
- Det betyr at forelesningene er å betrakte som pensum, også der de går ut over rammene til læreboka.
- Alt stoff som er eksamensrelevant vil man finne i læreboka eller i forelesningsnotatene som legges ut på nettet.

Innledning til rekursjon og induksjon

- Læreboka behandler for det meste rekursjon og induksjon over de naturlige tallene \mathbb{N} .

- I en IT-sammenheng finnes det andre induktivt konstruerte mengder hvor tilsvarende metoder har mening.
- Vi skal etterhvert se på noen generelle og spesielle eksempler av interesse for IT.
- Vi skal imidlertid først se på rekursjon i en begrenset, men viktig, forstand.

Rekursjon

Eksempel.

- Vi definerer en funksjon $f : \mathbb{N} \rightarrow \mathbb{N}$ ved
 1. $f(1) = 2$
 2. $f(n + 1) = 2^{f(n)}$ for alle n .
- Vi har ikke definert f ved en formel, så er f veldefinert?

Rekursjon

Eksempel (Fortsatt).

- En test kan jo være om vi er i stand til å skrive et program for $f!$
- Vi kan oppfatte punktene 1. og 2. på forrige side som en spesifikasjon.
- Vi har tidligere sett hvordan vi kan finne en pseudokode for $g(z) = 2^z$
- Det betyr at vi kan bruke en instruksjon på formen

$$z \leftarrow 2^y$$

med vissheten om at vi kan erstatte den ene linjen med en pseudokode.

- Da er det lett å lage en pseudokode for f :

Rekursjon

Eksempel (Fortsatt).

1. *Input* x [$x \in \mathbb{N}$]
2. $z \leftarrow 2$
3. $i \leftarrow 1$
4. **While** $i < x$ **do**
 - 4.1 $i \leftarrow i + 1$
 - 4.2 $z \leftarrow 2^z$
5. *Output* z

Vi kaller $f(x)$ verdien på 2^{er} -tårnet av høyde x .

Rekursjon

Eksempel.

- Vårt neste eksempel er en funksjon som brukes mye i matematikk og i sannsynlighetsregning,

$$n \mapsto n!,$$

eller fakultetsfunksjonen.

- Vi kan bruke omtrent samme formatet som i forrige eksempel:

1. $1! = 1$

2. $(n + 1)! = n! \cdot (n + 1)$ for alle $n \in \mathbb{N}$.

- Vi kan nærmest kopiere pseudokoden fra forrige eksempel, og får følgende algoritme for beregning av $n!$:

Rekursjon

Eksempel (Fortsatt).

1. *Input* x [$x \in \mathbb{N}$]

2. $z \leftarrow 1$

3. $i \leftarrow 1$

4. **While** $i < x$ **do**

4.1 $i \leftarrow i + 1$

4.2 $z \leftarrow z \cdot (i)$

5. *Output* z

Rekursjon

- Læreboka tar utgangspunkt i tallfølger, mens vi tar utgangspunkt i funksjoner.
- Det er i prinsippet ingen forskjell mellom en uendelig tallfølge og en funksjon definert på \mathbb{N}
- Tallfølgen

$$1, 2, 6, 24, 120, 720, \dots$$

er bare en annen måte å skrive fakultetsfunksjonen på.

- Hvorvidt man i konkrete tilfeller bruker tallfølger eller funksjoner, avhenger av hva som er pedagogisk mest forstandig for anledningen.

Rekursjon

- Kan vi gi en bedre begrunnelse for at de to funksjonene vi har sett på er veldefinerte enn at vi kan finne pseudokoder for dem?

- Svaret er selvfølgelig *JA*.
- Vi kan nå alle naturlige tall ved å
 1. Starte med 1
 2. Legge til 1 så mange ganger som nødvendig.
- Hvis vi da definerer en funksjon f ved å bestemme
 1. hva $f(1)$ er
 2. hvordan $f(n + 1)$ avhenger av $f(n)$ og n
 har vi bestemt $f(n)$ for alle n .
- Vi kan oppfatte en konkretisering av punktene 1 og 2 over som en spesifikasjon.
- Vi skal se på et eksempel i detalj:

Rekursjon

Eksempel.

Vi definerer funksjonen $f(n, m)$ ved rekursjon på n ved

1. $f(1, m) = 2m - 1$
 2. $f(n + 1, m) = 2f(n, m) - 1$
- Med tilstrekkelig tålmodighet kan vi finne et uttrykk for $f(n, m)$ for hver enkelt n ved:

Rekursjon

Eksempel.

1. $f(1, m) = 2m - 1$
 2. $f(2, m) = 2f(1, m) - 1 = 2(2m - 1) - 1 = 4m - 3$
 3. $f(3, m) = 2f(2, m) - 1 = 2(4m - 3) - 1 = 8m - 7$
 4. $f(4, m) = 2f(3, m) - 1 = 2(8m - 7) - 1 = 16m - 15$
- ...

Vi ser at vi kan gjøre listen av utregninger så lang vi vil, så $f(n, m)$ er definert for alle n og m .

En annen sak er om vi kan vise den formelen som ser ut til å peke seg ut.

Da vil vi få bruk for induksjonsbevis.

Rekursjon

- Læreboka har brukt **For**-løkker der vi har brukt **While**-løkker.
- Forskjellen er kosmetisk.

- Det viktige er at vi bruker en løkke til å fange opp formatet
 1. $g(1) = a$
 2. $g(n + 1) = f(g(n), n)$
- og at vi har en standard overgang fra en pseudokode for f til en pseudokode for g .
- Vi sier at g er definert fra a og f ved rekursjon.
- Vi beskriver den generelle **Før-løkka** på neste side:

Rekursjon

1. *Input* $n [n \in \mathbb{N}]$
2. $x \leftarrow a$
3. **For** $m = 2$ **to** n **do**
 - 3.1 $x \leftarrow f(x, m)$
4. *Output* x

Merk.

Vi sier at klassen av funksjoner programmerbare via en pseudokode er lukket under *definisjoner ved rekursjon*.

Rekursjon

Oppgave.

Betrakt følgende pseudokode, hvor det inngår en rekursiv definisjon:

1. *Input* $n [n \in \mathbb{N}]$
2. $x \leftarrow 1$
3. $y \leftarrow 1$
4. $z \leftarrow 1$
5. **For** $m = 2$ **to** n **do**
 - 5.1 $y \leftarrow y + 1$
 - 5.2 **For** $k = 1$ **to** y **do**
 - 5.2.1 $z \leftarrow z + 1$
 - 5.2.2 $x \leftarrow x + z$
6. *Output* x

Rekursjon

Oppgave (Fortsatt).

- Følg beregningen og finn verdien på output for $n = 1$, $n = 2$, $n = 3$ og $n = 4$.
- Hvordan tror du denne følgen fortsetter?
- Vil beregningen stoppe uansett hvilket naturlig tall n vi starter med?

Rekursjon

- Til nå har vi bare sett på funksjoner fra \mathbb{N}_0 til \mathbb{N}_0 definert ved rekursjon.
- Filosofien bak hvorfor rekursive definisjoner gir mening gir oss også muligheten til å betrakte andre definisjonsområder:

Rekursjon

Eksempel.

La $f(2) = 1$

Hvis $n \geq 2$ definerer vi $f(n+1)$ ved

- $f(n+1) = f(n)$ hvis n ikke er et primtall.
- $f(n+1) = f(n) + 1$ hvis n er et primtall.

Da er $f(n)$ definert for alle tall $n \geq 2$ og forteller oss hvor mange primtall det finnes $\leq n$.

Rekursjon

- Foreløpig gir det ikke mening å bruke rekursjon til å definere funksjoner med definisjonsområder som ikke er \mathbb{N} , \mathbb{N}_0 eller $\{n \in \mathbb{N} : n \geq k\}$ for en k .
- Det er imidlertid ingen grunn til at verdiområdet skal bestå av tall, noe vårt neste eksempel vil vise:

Rekursjon

Eksempel.

- Vi har en klassisk definisjon av regningsart R_n nummer n :
- $R_1(x, y) = x + y$
- R_2 defineres rekursivt ved
 - $R_2(0, y) = 0$
 - $R_2(x + 1, y) = R_1(R_2(x, y), y)$
- Hvis $n \geq 2$ og R_n er definert, definerer vi R_{n+1} rekursivt ved

- $R_{n+1}(0, y) = 1$
- $R_{n+1}(x + 1, y) = R_n(R_{n+1}(x, y), y)$.

Vi ser at R_1 er addisjon, R_2 er multiplikasjon, R_3 er eksponensiering osv.

Induksjonsbevis

Eksempel.

- La oss gå tilbake til den rekursive definisjonen

1. $f(1, m) = 2m - 1$

2. $f(n + 1, m) = 2f(n, m) - 1$

hvor det er naturlig å gjette på at

$$f(n, m) = 2^n \cdot m - (2^n - 1).$$

- Vi har sett at denne formelen stemmer for $n = 1$, $n = 2$, $n = 3$ og $n = 4$ da vi regnet ut

Induksjonsbevis

Eksempel (Fortsatt).

1. $f(1, m) = 2m - 1$

2. $f(2, m) = 2f(1, m) - 1 = 2(2m - 1) - 1 = 4m - 3$

3. $f(3, m) = 2f(2, m) - 1 = 2(4m - 3) - 1 = 8m - 7$

4. $f(4, m) = 2f(3, m) - 1 = 2(8m - 7) - 1 = 16m - 15$

- Hvis vi nu prøver å se om formelen stemmer for $n = 5$ på en slik måte at vi forhåpentligvis finner en forklaring, kan vi regne som følger:

Induksjonsbevis

Eksempel (Fortsatt).

$$f(5, m) = 2f(4, m) - 1 = 2(2^4 m - (2^4 - 1)) - 1 =$$

$$2 \cdot 2^4 m - 2(2^4 - 1) - 1 =$$

$$= 2^5 m - 2^5 + 2 - 1 = 2^5 m - (2^5 - 1).$$

Induksjonsbevis

Eksempel (Fortsatt).

- I denne utregningen har vi bare brukt at $5 = 4 + 1$
- Vi kunne erstattet 4 med en vilkårlig n og 5 med $n + 1$, og fått utregningen

$$\begin{aligned} f(n + 1, m) &= 2f(n, m) - 1 = 2(2^n m - (2^n - 1)) - 1 = \\ &= 2 \cdot 2^n m - 2(2^n - 1) - 1 = \\ &= 2^{n+1} m - 2^{n+1} + 2 - 1 = 2^{n+1} m - (2^{n+1} - 1). \end{aligned}$$

- Dermed har vi gitt det som kalles et induksjonsbevis for at formelen vår er riktig.

Induksjonsbevis

- Hvorfor kan vi betrakte argumentet over som et bevis for at formelen holder for alle n ?
- Årsaken er at vi nå vet at vi ved direkte utregning kan bevise formelen hver gang noen gir oss en verdi for n , for eksempel $n = 8$.
- Vi vet nå at formelen holder for $n = 5$ og vi vet at siden den holder for $n = 5$ må den holde for $n = 6$.
- Utregningen vår gir imidlertid at formelen også må holde for $n = 7$ og deretter for $n = 8$.
- Siden vi vet at vi med utholdenhet kan fortsette å tenke slik så langt noen kunne ønske, vet vi at formelen vår må holde for alle n .

Induksjonsbevis

- Den metoden vi har brukt til å bevise en påstand for alle naturlige tall på kalles som sagt induksjonsbevis.
- I sin enkleste form kan induksjonsbevis formuleres som:

Definisjon.

La $P(n)$ være et predikat med en variabel n for et element i \mathbb{N} .

Anta at vi kan bevise

1. $P(1)$
2. $\forall n(P(n) \rightarrow P(n + 1))$

Da kan vi konkludere $\forall n P(n)$.

Denne måten å bevise $\forall n P(n)$ på kalles induksjon.

Forelesning 15

Induksjon og rekursjon, rekurenslikninger

Dag Normann - 3. mars 2008

Rekursjon og induksjon

- Onsdag ga vi endel eksempler på rekursive definisjoner og vi forklarte hva vi mener med induksjonsbevis.
- Vi kommer til å fortsette i dag med å gi eksempler på begge deler.
- Induksjonsbevis er et effektivt matematisk virkemiddel.
- Våre eksempler vil ofte gå ut på å vise at en formel for det generelle leddet i en følge som vi har definert ved rekursjon er riktig.
- I flere eksempler vil den naturlige gangen være

Problem \rightarrow rekursjon \rightarrow formel \rightarrow induksjonsbevis.

Induksjonsbevis

Eksempel.

- Definer
 - $f(1) = 1$
 - $f(n + 1) = 3f(n) + 1$ for alle $n \in \mathbb{N}$.

Da er $f(1) = 1$, $f(2) = 3 \cdot 1 + 1 = 4$, $f(3) = 3 \cdot 4 + 1 = 13$ og $f(4) = 3 \cdot 13 + 1 = 40$.

Vi kan fortsette å regne ut $f(5)$, $f(6)$, $f(7)$ osv. ettersom f er definert ved rekursjon.

Induksjonsbevis

Eksempel (Fortsatt).

- Definer
 - $g(1) = 1$
 - $g(n + 1) = g(n) + 3^n$

Da er $g(1) = 1$, $g(2) = 1 + 3 = 4$, $g(3) = 4 + 3^2 = 13$ og $g(4) = 13 + 3^3 = 40$.

Vi kan fortsette å regne ut $g(5)$, $g(6)$ osv. , og vil finne ut at så langt vi kan se vil $f(n) = g(n)$.

Induksjonsbevis

Eksempel (Fortsatt).

- Det er da naturlig å gjette på at $f(n) = g(n)$ for alle n .
- For å vise det, kan vi prøve å vise at de to rekursive definisjonene er de samme, men vi ser jo at rekursjonskrittet i de to definisjonene ikke likner på hverandre.
- Vi ser at $g(n)$ er summen i en endelig geometrisk rekke

$$g(n) = 1 + 3 + 3^2 + \dots + 3^{n-1}.$$

- En slik rekke har en kjent sum

$$g(n) = \frac{3^n - 1}{3 - 1} = \frac{3^n - 1}{2}.$$

Induksjonsbevis

Eksempel (Fortsatt).

- Siden vi sikkert ikke husker hvordan vi kom frem til denne formelen, og siden vi innfører induksjonsbevis i disse forelesningene, viser vi formelen ved induksjon: Setter vi $n = 1$ inn i formelen, får vi

$$\frac{3^n - 1}{2} = \frac{3 - 1}{2} = 1 = g(1)$$

så formelen stemmer for $n = 1$.

- Anta at formelen stemmer for et tall n , det vil si at

$$g(n) = \frac{3^n - 1}{2}.$$

Induksjonsbevis

Eksempel (Fortsatt).

Da er

$$\begin{aligned} g(n+1) &= g(n) + 3^n = \frac{3^n - 1}{2} + 3^n \\ &= \frac{3^n - 1 + 2 \cdot 3^n}{2} = \frac{3 \cdot 3^n - 1}{2} = \frac{3^{n+1} - 1}{2} \end{aligned}$$

som viser at formelen også holder for $g(n+1)$.

Induksjonsbevis

Eksempel (Fortsatt).

Hvis $P(n)$ er utsagnet

$$g(n) = \frac{3^n - 1}{2}$$

har vi vist

$P(1)$ for seg

1. $P(1) \rightarrow P(2)$
2. $P(2) \rightarrow P(3)$
3. $P(3) \rightarrow P(4)$
- ...

under ett som spesialtilfeller av $P(n) \rightarrow P(n + 1)$.

Induksjonsbevis

Eksempel (Fortsatt).

Da er eksempelvis $P(17)$ en tautologisk konsekvens av alt det vi har bevist.

Prinsippet bak induksjonsbevis er at vi da vet med sikkerhet at $P(n)$ holder for alle n .

Induksjonsbevis

Eksempel (Fortsatt).

La nå $Q(n)$ være påstanden

$$f(n) = \frac{3^n - 1}{2}.$$

Vi skal se at vi også kan vise $\forall n Q(n)$ ved induksjon.

Det vil følge at f og g er de samme funksjonene, eller de samme følgene hvis man ønsker å se på det på den måten.

Induksjonsbevis

Eksempel (Fortsatt).

- Induksjonstarten er grei, siden $f(1) = 1 = g(1)$ og vi vet at formelen holder for g .

- Anta så at

$$f(n) = \frac{3^n - 1}{2}.$$

- Da er

$$f(n+1) = 3 \cdot f(n) + 1 = 3 \cdot \frac{3^n - 1}{2} + 1$$

$$= \frac{3^{n+1} - 3}{2} + \frac{2}{2} = \frac{3^{n+1} - 1}{2}.$$

- Dette viser induksjonskrittet, hvis $Q(n)$ holder, så vil $Q(n+1)$ holde.
- Konklusjonen er at $f(n) = g(n) = \frac{3^n - 1}{2}$ for alle n .

Induksjonsbevis

I læreboka presenteres et klassisk eksempel på bruk av induksjon:

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}.$$

Dette en formel man finner igjen i direkte eller beslektet form i mange viktige sammenhenger.

Eksempelvis er det antallet oppgjør i en enkel serie med n lag.

Vi skal se på noen andre, delvis beslektede eksempler:

Induksjonsbevis

Eksempel.

- La $f(n)$ være summen av de første n oddetallene, det vil si at

$$f(n) = \sum_{i=1}^n (2i - 1)$$

Da er $f(n) = n^2$,

- Vi skal gi et induksjonsbevis.

Induksjonsbevis

Eksempel (Fortsatt).

- For å vise starten på induksjonen regner vi ut

$$f(1) = \sum_{i=1}^1 (2i - 1) = 1 = 1^2.$$

- Deretter må vi gjennomføre induksjonskrittet:
Anta at $f(n) = n^2$ for en n .
Da er $f(n + 1) = f(n) + 2n + 1 = n^2 + 2n + 1 = (n + 1)^2$.
- Ettersom vi nå har vist både induksjonstarten og induksjonskrittet, følger påstanden ved induksjon.

Induksjonsbevis

Eksempel.

- Hvis vi trekker en rett linje gjennom planet, deler vi planet i to.
- Hvis vi trekker en ny linje gjennom planet, deler disse to linjene planet i fire deler.
- Hvis vi prøver oss med tre linjer, greier vi ikke å dele planet i mer enn syv deler, og bruker vi fire linjer greier vi maksimalt å dele planet i 11 deler.
- Kan vi finne en formel for hvor mange felter vi maksimalt kan dele planet i ved hjelp av n linjer?

Induksjonsbevis

Eksempel (Fortsatt).

- La $F(n)$ være antall felter vi kan dele planet opp i ved å bruke n rette linjer.
- Da er $F(1) = 2$.
- Selv om vi ikke kjenner $F(n)$ kan vi uttrykke $F(n + 1)$ ved hjelp av $F(n)$:
- La l_1, \dots, l_n, l_{n+1} være $n+1$ rette linjer slik at l_1, \dots, l_n deler planet opp i $F(n)$ forskjellige felter.
- Den siste linjen l_{n+1} skjærer hver av de andre linjene høyst en gang, så vi får maksimalt n nye skjæringspunkter.

Induksjonsbevis

Eksempel (Fortsatt).

- Skjæringspunktene deler l_{n+1} opp i høyst $n + 1$ linjestykker, og hvert av disse linjestykkene deler et av de gamle feltene i to.
- Det betyr at vi får maksimalt $n + 1$ nye felter.
- Da er $F(n + 1) = F(n) + n + 1$
- Den neste jobben blir å finne en formel for $F(n)$ og så vise den ved induksjon.

- Denne typen formler finner man ofte gjennom prøving og feiling basert på erfaring.

Induksjonsbevis

Eksempel (Fortsatt).

- Vi påstår at $F(n) = 1 + \frac{n(n+1)}{2}$ og vil vise det ved induksjon:

- Induksjonen starter med $n = 1$:

$$1 + \frac{1(1+1)}{2} = 1 + 1 = 2 = F(1).$$

- La oss så gjennomføre induksjonskrittet:
- Anta at

$$F(n) = 1 + \frac{n(n+1)}{2}$$

Induksjonsbevis

Eksempel (Fortsatt).

- Da er

$$\begin{aligned} F(n+1) &= F(n) + n + 1 = 1 + \frac{n(n+1)}{2} + (n+1) \\ &= 1 + \frac{n(n+1) + 2(n+1)}{2} = 1 + \frac{(n+2)(n+1)}{2}. \end{aligned}$$

- Skal vi være pedantiske kan vi skrive dette om til

$$1 + \frac{(n+1)((n+1)+1)}{2}.$$

- Induksjonskrittet er gjennomført, så påstanden er bevist.

Induksjonsbevis

Oppgave.

- Vi vet at vi kan dele planet opp i to felter ved hjelp av en sirkel.
- Vi vet at to sirkler kan skjære hverandre i to punkter
- Vi vet at $2n$ punkter vil dele en sirkel opp i $2n$ buestykker.
- Bruk dette til å definere funksjonen $G(n)$ ved rekursjon, hvor $G(n)$ er antall områder vi kan dele planet opp i ved hjelp av n sirkler.
- Foreslå en formel for $G(n)$ og se om du kan vise den ved induksjon.

Hvorfor forteller svaret på denne oppgaven oss at Venn-diagrammer er uegnet til å studere Boolske kombinasjoner av mange mengder?

Induksjonsbevis

Eksempel.

- Enkelte regneoperasjoner tar lengere tid jo større input er.
- Det kan være av interesse å finne ut hvor mange "regneskritt" en oppgave krever, avhengig av hvor stort input er.
- Eksempelvis kan vi prøve å finne ut av hvor mange operasjoner som kreves for å utføre sorteringsalgoritmer i
 - Verste tilfelle
 - I gjennomsnitt

Induksjonsbevis

Eksempel (Fortsatt).

- Vanligvis greier man seg med omtrentlige verdier, men ved behov kan man bruke rekursjon og induksjon til å finne nøyaktige svar.
- Vi kan sortere elementene i en liste ved systematisk å bytte om på naboer som ligger i gal rekkefølge.
- La $S(n)$ være det maksimale antall slike bytter vi må foreta oss for å sortere en liste.
- Vi ser at $S(1) = 0$
- Hvis listen kommer i fullstendig gal rekkefølge, må alle objektene i listen bytte plass med alle andre.

Induksjonsbevis

Eksempel (Fortsatt).

- Antall bytter som da trengs for å sortere $n + 1$ objekter er $S(n + 1) = n + S(n)$, ettersom vi kan risikere at vi må flytte siste objekt i listen til førsteplass (n bytter) og deretter sortere resten av listen ($S(n)$ bytter.)
- Vi ser ved induksjon at $S(n) = \frac{(n-1)n}{2}$.
- Beviset følger ved samme type utregning i induksjonsskrittet som for forrige eksempel, og vi tar det på tavlen (eller som øvelse for de som leser/repeterer denne teksten).

Induksjonsbevis

Merk.

- Det forrige eksemplet er ikke helt realistisk, enhver sorteringsalgoritme vil innebære at man foretar en rekke sammenlikninger og skifte av plasser.
- Hvis vi skal analysere hvor tidkrevende en algoritme kan være, må vi vite hvor mange regneskritt som kreves, og hvor lang tid hvert enkelt skritt tar.
- Induksjonsbevis kan inngå som en del av beviset for at en regneprosess kan utføres raskt, eventuelt for at den tar for lang tid.

Induksjonsbevis

Eksempel.

- Vi minner om definisjonen av binomialkoeffisienten

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

- Formelen

$$\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$$

kan bekreftes ved enkel regning.

- Den enkle regningen ble gjennomført på tavlen.

Induksjonsbevis

Eksempel (Fortsatt).

- På skolen lærer man at

$$\binom{n}{k}$$

uttrykker på hvor mange måter man kan velge ut k objekter fra en mengde med n objekter på, når $k \leq n$.

- Det er ikke alltid så lett å få med seg begrunnelsen for dette.
- Et alternativ kan være å bruke induksjon.

Induksjonsbevis

Eksempel (Fortsatt).

- Vi starter med tilfellet $n = 1$.
- Da er $k = 1$, og det finnes bare en måte å velge ut ett element fra en mengde på ett element. Binomialkoeffisienten er i dette tilfellet 1, så påstanden holder.
- Induksjonstarten er i boks.

Induksjonsbevis

Eksempel (Fortsatt).

- Anta så at formelen holder for n og at vi skal finne ut av på hvor mange måter vi kan plukke k elementer ut av en mengde $\{a_1, \dots, a_{n+1}\}$ på.
- Hvis $k = n + 1$, finnes det nøyaktig en måte, og

$$1 = \binom{n+1}{n+1}.$$

Induksjonsbevis

Eksempel (Fortsatt).

- Hvis $k < n + 1$, ser vi på to tilfeller:
 1. a_{n+1} er med i den mengden vi plukker ut.
 2. a_{n+1} er ikke med i den mengden vi plukker ut.
- I det første tilfellet må vi plukke ut $k - 1$ elementer fra

$$\{a_1, \dots, a_n\},$$

og det kan vi gjøre på

$$\binom{n}{k-1}$$

måter.

Induksjonsbevis

Eksempel (Fortsatt).

- I det andre tilfellet må vi plukke ut k elementer fra

$$\{a_1, \dots, a_n\},$$

og det kan vi gjøre på

$$\binom{n}{k}$$

måter.

- Summen er da

$$\binom{n+1}{k}.$$

som angir det totale antall måter vi kan plukke ut k elementer fra en mengde med n elementer på.

Induksjonsbevis

Eksempel (Fortsatt).

- Induksjonskrittet sier at hvis binomialkoeffisientene $\binom{n}{k}$ forteller oss, for alle $k \leq n$, hvor mange forskjellige delmengder med k elementer det finnes av en mengde med n elementer, så vil koeffisientene $\binom{n+1}{k}$ fortelle oss det samme for mengder med $n+1$ elementer.
- Vi kan merke oss at for å vise induksjonskrittet for en k trenger vi induksjonsantagelsen både for k og for $k-1$.

Induksjonsbevis

- Både rekursjon og induksjon er mere generelle fenomener enn det vi har gitt inntrykk av her.
- Dette skal vi komme tilbake til, både ved å se på rekurrenslikninger og på rekursjon og induksjon over andre matematiske strukturer enn \mathbb{N} eller \mathbb{N}_0 .
- Først skal vi imidlertid se på en logikers forklaring på sammenhengen mellom induksjon og rekursjon:
- Anta at vi i utgangspunktet ikke har lært om induksjonsbevis.
- Anta videre at $P(k)$ er et predikat og at vi har bevis for
 1. Induksjonstarten $P(1)$
 2. Induksjonskrittet $P(k) \rightarrow P(k+1)$ hvor k er en variabel.

Induksjonsbevis

- Ved *rekursjon* kan vi da konstruere et bevis $B(n)$ for $P(n)$ for enhver n ved
 1. La $B(1)$ være beviset vi har for $P(1)$
 2. La $B(n+1)$ være bygget opp av $B(n)$, beviset vi får for $P(n) \rightarrow P(n+1)$ ved å sette inn n for k i beviset for induksjonskrittet, og bruk av den utsagnslogiske regelen om at fra A og $A \rightarrow B$ kan vi slutte B .

- Når vi vet at vi kan konstruere enkeltbevis for hver $P(n)$, kan vi rasjonalisere virksomheten vår og kalle dette et bevis for $\forall n P(n)$.

Rekurrens

- Det i enhver sammenheng mest brukte eksemplet på rekurrens er definisjonen av Fibonacci-tallene:
 - $F(1) = 1$
 - $F(2) = 1$
 - $F(n + 2) = F(n + 1) + F(n)$ for alle $n \in \mathbb{N}$.
- Vi ser at denne tallfølgen også er fullstendig bestemt, selv om definisjonen ikke helt følger formatet til definisjoner ved rekursjon.
 1. $F(1) = 1$
 2. $F(2) = 1$
 3. $F(3) = F(2) + F(1) = 1 + 1 = 2$
 4. $F(4) = F(3) + F(2) = 2 + 1 = 3$
 5. $F(5) = F(4) + F(3) = 3 + 2 = 5$
 - ...

Rekurrens

- Det er ingen grenser for hvor langt vi kan fortsette:

$$F(6) = 8, F(7) = 13, F(8) = 21, F(9) = 34, \dots$$
- Spørsmålet er om vi kan finne en eksplisit formel for $F(n)$, og helst om vi kan basere dette på en generell forståelse.
- Dette skal vi komme tilbake til.
Vi skal se på et par andre eksempler først.

Rekurrens

Eksempel.

- Likningen

$$F(n + 2) = F(n + 1) + 2F(n)$$

bestemmer ikke tallfølgen $F(1), F(2), F(3), \dots$ fullstendig, men hver gang vi bestemmer oss for hva $F(1)$ og $F(2)$ skal være, blir følgen bestemt ved rekurrens.

- En slik likning kaller vi en Rekurrenslikning.
- Ved direkte regning kan vi se at $F_1(n) = 2^n$ og $F_2(n) = (-1)^n$ begge tilfredstiller likningen:
 - $2^{n+1} + 2 \cdot 2^n = 2^{n+1} + 2^{n+1} = 2^{n+2}$
 - $(-1)^{n+1} + 2 \cdot (-1)^n = (-1)^n(-1 + 2) = (-1)^n(-1)^2 = (-1)^{n+2}$

Rekurrens

Eksempel (Fortsatt).

- Hvis nå A og B er to reelle tall, ser vi at vi også har at

$$F_3(n) = A \cdot 2^n + B \cdot (-1)^n$$

er en løsning.

- Er det nå noen grunn til å lete etter flere løsninger?
- Svaret er *NEI*, for vi vet at hvis vi bestemmer $F(1) = a$ og $F(2) = b$, har vi bestemt følgen fullstendig.
- Likningene $a = A \cdot 2^1 + B \cdot (-1)^1$ og $b = A \cdot 2^2 + B \cdot (-1)^2$ vil bestemme A og B , slik at løsningen i et konkret tilfelle er en av de vi har sett på.

Rekurrens

Eksempel (Fortsatt).

Løsningene er

-

$$A = \frac{a + b}{6}$$

-

$$B = \frac{2b - a}{3}$$

- Det neste spørsmålet er da selvfølgelig hvordan vi fant på å prøve potenser av 2 og -1 .

Forelesning 16

Rekurrenslikninger

Dag Normann - 5. mars 2008

Rekurrens

INGEN PLENUMSREGNING 6/3 og 27/3

Rekurrens

- Mandag ga vi en rekke eksempler på bruk av induksjonsbevis og rekursivt definerte funksjoner.
- I et av eksemplene definerte vi binomialkoeffisientene og viste en viktig egenskap til binomialkoeffisienter ved induksjon.
- Både definisjonen og den viste egenskapen er å betrakte som pensum.
- Det siste kvarteret snakket vi om rekurrens.
- Etter forelesningen spurte en av studentene om hva forskjellen mellom rekurrenslikninger og differenslikninger er.
- Svaret er at det ikke er noen forskjell, og at de som har lært teori om differenslikninger kan overføre det til rekurrenslikninger.
- Vi skal nå fortsette med noen eksempler som tilnærming til en generell metode for å løse rekurrenslikninger.

Rekurrens

Eksempel.

- Vi skal lete etter løsninger av rekurrenslikningen

$$F(n+2) = 5F(n+1) - 6F(n)$$

- Vi viser først ved regning at $F_1(n) = 2^n$ og $F_2(n) = 3^n$ er løsninger:
- $5 \cdot 2^{n+1} - 6 \cdot 2^n = 2^n(5 \cdot 2 - 6) = 2^n \cdot 2^2 = 2^{n+2}$
- $5 \cdot 3^{n+1} - 6 \cdot 3^n = 3^n(5 \cdot 3 - 6) = 3^n \cdot 3^2 = 3^{n+2}$
- Det som gjør at denne utregningen fører frem er at 2 og 3 er løsninger av likningen

$$x^2 = 5x - 6$$

og det er de eneste løsningene.

- Det betyr at man må kunne løse 2. gradsligninger for å kunne løse slike rekurrenslikninger.

Rekurrens

I stedet for å skrive at

$$F(n+2) = F(n+1) + 2F(n)$$

skal gjelde for alle $n \in \mathbb{N}$

kan vi skrive at

$$F(n) - F(n-1) - 2F(n-2) = 0$$

for alle $n \geq 2$,

og i stedet for å skrive at

$$F(n+2) = 5F(n+1) - 6F(n)$$

for alle $n \in \mathbb{N}$

kan vi skrive at

$$F(n) - 5F(n-1) + 6F(n-2) = 0$$

for alle $n \geq 2$.

Rekurrens

Dette er strengt tatt uendelig mange likninger i uendelig mange variable $F(n)$, men det gir mening å snakke om løsningsmengden til dette likningsettet.

Vi vil nå etablere den terminologien vi skal bruke i fortsettelsen, og vise hvordan vi kan finne løsningsmengden til slike *rekurrenslikninger*.

Rekurrens

Definisjon.

- En 2. ordens lineær homogen rekurrenslikning er en *funksjonslikning* på formen

$$at(n) + bt(n-1) + ct(n-2) = 0.$$

- En *følge* $F: \mathbb{N} \rightarrow \mathbb{R}$ er en løsning av rekurrenslikningen hvis $aF(n) + bF(n-1) + cF(n-2) = 0$ for alle $n \geq 2$.
- Hvis verdiene for $t(1)$ og/eller $t(2)$ er angitt i tillegg, kalles dette initialbetingelser.
- En løsning må da tilfredstille disse betingelsene.

Rekurrens

Eksempel.

- Fibonacci-følgen er bestemt som den eneste løsningen $F: \mathbb{N} \rightarrow \mathbb{N}$ av

$$t(n) - t(n-1) - t(n-2) = 0$$

med initialbetingelser $t(1) = t(2) = 1$

- Vi skal finne en formel for $F(n)$ når vi har utviklet metoden for det.

Rekurrens

Merk.

- Vi kaller rekurrenslikningen lineær fordi likningens venstre side er en lineær kombinasjon av $t(n)$, $t(n-1)$ og $t(n-2)$
Eksempelvis vil ikke $t(n) - (t(n-1))^2 - t(n-2)$ være lineær.
- Vi kaller likningen homogen fordi vi ikke har noe ledd som bare avhenger av n .
Eksempelvis er ikke $t(n) - t(n-1) + 0 \cdot t(n-2) + n = 0$ homogen.
- Likningen er 2. ordens fordi verdien av $t(n)$ avhenger av to verdier $t(n-1)$ og $t(n-2)$.
 $t(n) - 2t(n-1) = 0$ er 1. ordens, mens $t(n) - t(n-1) + t(n-2) - t(n-3) = 0$ er 3. ordens.

Rekurrens

Merk (Fortsatt).

- Alt vi sier vil kunne gjøres gjeldende for 1. ordens og 3. ordens homogene, lineære likninger også, men vi skal konsentrere oss om de 2. ordens likningene.
- For enkelthets skyld vil vi bruke betegnelsen rekurrenslikning i betydningen 2. ordens lineær homogen rekurrenslikning, ettersom vi vil begrense oss til denne typen rekurrenslikninger.

Rekurrens

- Da vi analyserte mulige løsninger av likningen

$$F(n+2) = 5F(n+1) - 6F(n)$$

kommenterte vi at vi utnyttet at 3 og 2 er løsninger i likningen

$$x^2 = 5x - 6$$

da vi viste at $F_1(n) = 3^n$ og $F_2(n) = 2^n$ er løsninger av rekurrenslikningen.

- Denne innsikten skal vi utvikle til en fullstendig innsikt i hvilke løsninger vi har:

Rekurrens

Definisjon.

Hvis

$$at(n) + bt(n - 1) + ct(n - 2) = 0$$

er en rekurrenslikning, kalles

$$ax^2 + bx + c = 0$$

for den karakteristiske likningen til rekurrenslikningen.

Rekurrens**Teorem.**

La

$$at(n) + bt(n - 1) + ct(n - 2) = 0$$

være en rekurrenslikning og la $r \in \mathbb{R}$.

Da er $F_r(n) = r^n$ en løsning av rekurrenslikningen hvis og bare hvis r er en løsning av den *karakteristiske* likningen.

Rekurrens**Bevis.**

Anta at r er en løsning av den karakteristiske likningen.

Vi setter F_r inn i likningene, og får

$$ar^n + br^{n-1} + cr^{n-2} = r^{n-2}(ar^2 + br + c) = 0,$$

det siste fordi r er løsning av den karakteristiske likningen.

Anta så at $F_r(n) = r^n$ løser rekurrenslikningen.

Setter vi inn for $n = 2$ får vi spesielt

$$ar^2 + br^1 + cr^0 = 0$$

som akkurat sier at $ar^2 + br + c = 0$.

Rekurrens

Merk.

- Vi kunne ha problematisert tilfellet hvor $r = 0$.
- Argumentet virker bare under antagelsen om at $0^0 = 1$, $0^n = 0$ når $n > 0$.
- Vi får bare $r = 0$ som løsning av den karakteristiske likningen når $c = 0$, hvilket egentlig betyr at vi har en 1. ordens rekurrenslikning.
- Da er alle løsningene på formen $F(n) = A \cdot r^n$, hvor r er den andre løsningen av den karakteristiske likningen.
- Konstantfølgen 0 er alltid en løsning av en rekurrenslikning uten initialbetingelser.

Rekurrens

Dette resultatet forteller oss at vi ofte kan finne to løsninger av en rekurrenslikning.

Hvis løsningene av den karakteristiske likningen inneholder kvadratrott-tegn, vil de eksakte formlene for løsningene gjøre det samme.

Hvis løsningene av den karakteristiske likningen er komplekse tall, vil vi trenge komplekse tall for å beskrive løsningene av rekurrenslikningen.

Vi har imidlertid fortsatt bare to løsninger. Hvordan finner vi flere?

Setningen på neste side er et godt hjelpemiddel.

Rekurrens

Teorem.

La

$$at(n) + bt(n - 1) + ct(n - 2) = 0$$

være en rekurrenslikning ,

og la $F(n)$ og $G(n)$ være to løsninger.

La A og B være reelle tall.

Da er $H(n) = A \cdot F(n) + B \cdot G(n)$ også en løsning.

Rekurrens

Bevis.

- Det er bare å sette H inn i rekurrenslikningen og sjekke:

$$\begin{aligned} & a \cdot H(n) + b \cdot H(n - 1) + c \cdot H(n - 2) \\ &= a(A \cdot F(n) + B \cdot G(n)) + b(A \cdot F(n - 1) \end{aligned}$$

$$\begin{aligned}
& +B \cdot G(n-1) + c(A \cdot F(n-2) + B \cdot G(n-2)) \\
& = A \cdot (a \cdot F(n) + b \cdot F(n-1) + c \cdot F(n-2)) \\
& +B \cdot (a \cdot G(n) + b \cdot G(n-1) + c \cdot G(n-2)) \\
& = A \cdot 0 + B \cdot 0 = 0.
\end{aligned}$$

Rekurrens

Eksempel.

- La oss gå tilbake til likningen for Fibonacci-tallene:

$$t(n) - t(n-1) - t(n-2) = 0.$$

- Den karakteristiske likningen er

$$x^2 - x - 1 = 0$$

og ved abc-formelen har vi løsninger

$$r = \frac{1 \pm \sqrt{5}}{2}.$$

Rekurrens

Eksempel (Fortsatt).

Det betyr at vi bør lete etter en løsning på formen

$$F(n) = A\left(\frac{1 + \sqrt{5}}{2}\right)^n + B\left(\frac{1 - \sqrt{5}}{2}\right)^n.$$

Rekurrens

Eksempel (Fortsatt).

Vi vet at hvis vi finner A og B slik at initialbetingelsene $F(1) = F(2) = 1$ holder, så må vi ha funnet den eneste løsningen som finnes.

Det gir oss to *stygge* lineære likninger i de ukjente A og B

$$A \frac{1 + \sqrt{5}}{2} + B \frac{1 - \sqrt{5}}{2} = 1$$

$$A\left(\frac{1+\sqrt{5}}{2}\right)^2 + B\left(\frac{1-\sqrt{5}}{2}\right)^2 = 1$$

Den som vil løse disse likningene selv, bør lukke øynene på neste side, hvor vi gir løsningene uten mellomregning.

Rekurrens

Eksempel (Fortsatt).

Løsningene er

$$A = \frac{\sqrt{5}}{5}$$

og

$$B = -\frac{\sqrt{5}}{5}$$

så formelen for n'te Fibonaccitall $F(n)$ er, utrolig nok,

$$\frac{\sqrt{5}}{5} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$$

For hver verdi av n er altså dette et naturlig tall.

Rekurrens

- Hvis den karakteristiske likningen har to forskjellige løsninger r og s , vil alle løsningene av rekurrenslikningen være på formen

$$F(n) = A \cdot r^n + B \cdot s^n.$$

- Det er fordi hvis vi i tillegg bestemmer verdiene på $F(1)$ og $F(2)$, vil vi kunne løse likningsettet

$$A \cdot r + B \cdot s = F(1)$$

$$A \cdot r^2 + B \cdot s^2 = F(2).$$

- Dermed finner vi en løsning på formen $F(n) = Ar^n + Bs^n$ som oppfyller initialbetingelsene.
- Vi skal regne gjennom et type-eksempel.

Rekurrens

Eksempel.

- Anta at vi skal løse følgende oppgave:

a) Finn alle løsningene av rekurrenslikningen

$$t(n) - 2t(n-1) - 3t(n-2) = 0.$$

b) Finn løsningen fra a) som tilfredstiller initialbetingelsene $F(1) = 1$ og $F(2) = 2$.

- Det første vi må gjøre er å finne den karakteristiske likningen

$$x^2 - 2x - 3 = 0$$

og løse den:

Rekurrens

Eksempel (Fortsatt).

-

$$x = \frac{2 \pm \sqrt{2^2 + 4 \cdot 3}}{2} = \begin{cases} 3 \\ -1 \end{cases}$$

- Siden den karakteristiske likningen har to løsninger, vet vi at den generelle løsningen av rekurrenslikningen er $F(n) = A \cdot 3^n + B \cdot (-1)^n$.
- Dette løser a).

Rekurrens

Eksempel (Fortsatt).

- For å løse b), må vi bestemme A og B slik at initialbetingelsene holder.
- Det betyr at vi må løse likningene
 - $3A - B = 1$ ($n = 1$, $F(1) = 1$ var en betingelse.)
 - $9A + B = 2$ ($n = 2$, $F(2) = 2$ var en betingelse.)
- Legger vi sammen likningene, får vi $12A = 3$, så $A = \frac{1}{4}$ er en løsning.
- Setter vi denne løsningen inn i en av likningene og løser med hensyn på B, får vi $B = -\frac{1}{4}$
- Løsningen på oppgave b) er derfor

$$F(n) = \frac{1}{4}(3^n - (-1)^n).$$

Rekurrens

- I det forrige eksemplet så vi hvordan vi kan løse enhver rekurrenslikning hvor den karakteristiske likningen har to forskjellige løsninger.

- Hva gjør vi hvis det bare finnes en løsning?
- Fortsatt vil det være slik at hvis vi kan finne to “uavhengige” følger som løsninger, vil alle andre løsninger være kombinasjoner av disse.
- Hvis r er løsningen på den karakteristiske likningen, er fortsatt $F(n) = r^n$ en løsning av rekurrenslikningen.
- Målet vårt må derfor være å finne en annen løsning i tillegg.

Rekurrens

Eksempel.

- Betrakt rekurrenslikningen

$$t(n) - 4t(n-1) + 4t(n-2) = 0.$$

- Den karakteristiske likningen er

$$x^2 - 4x + 4 = 0.$$

- $r = 2$ er den eneste løsningen av den karakteristiske likningen.
- Da er $F(n) = 2^n$ en løsning av rekurrenslikningen.
- La oss gi to initialbetingelser for å se om det finnes noe mønster som antyder hvordan en annen løsning kan se ut:

Rekurrens

Eksempel (Fortsatt).

- $G(1) = 1$ og $G(2) = 0$:

Regning gir oss at

$$G(3) = -4, G(4) = -16, G(5) = -48 \text{ og } G(6) = -128$$

Hvis vi nå prøver å sette faktoren 2^n utenfor $G(n)$, får vi at

$$G(1) = 2^{-1} \cdot 2^1, G(2) = 0 \cdot 2^2, G(3) = -2^{-1} \cdot 2^3, G(4) = -2 \cdot 2^{-1} \cdot 2^4, G(5) = -3 \cdot 2^{-1} \cdot 2^5 \text{ og } G(6) = -4 \cdot 2^{-1} \cdot 2^6.$$

Det vil være naturlig å gjette på at $G(n) = (2-n) \cdot 2^{-1} \cdot 2^n$ er løsningen av rekurrenslikningen.

Vi skal se at det er tilfelle.

Rekurrens

Eksempel (Fortsatt).

- Hvis vi har gjettest riktig, vil $H(n) = n \cdot 2^n$ også være en løsning av rekurrenslikningen, siden H kan skrives som en kombinasjon av vårt forslag til $G(n)$ og av $F(n)$
- Omvendt, hvis $H(n)$ er en løsning av rekurrenslikningen, er $G(n)$ en kombinasjon av $F(n)$ og $H(n)$, så da er $G(n)$ en løsning av rekurrenslikningen (og den som oppfyller initialbetingelsene).
- I så fall er svaret så pent at det er verd et forsøk med et generelt bevis.

Rekurrens

Teorem.

La

$$at(n) + bt(n-1) + ct(n-2) = 0$$

være en rekurrenslikning hvor den karakteristiske likningen bare har en løsning r .

Da er $H(n) = nr^n$ en løsning av rekurrenslikningen.

Rekurrens

Bevis.

Løsningen av den karakteristiske likningen er

$$r = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Siden det bare er en løsning er $b^2 = 4ac$, fordi det som står under rottegnet må være null..

Vi setter $H(n)$ inn i likningen, og får

1. $anr^n + b(n-1)r^{n-1} + c(n-2)r^{n-2}$
2. $= r^{n-2}(anr^2 + bnr - br + cn - 2c)$
3. $= r^{n-2}(n(ar^2 + br + c) - br - 2c)$
4. $= -r^{n-2}(br + 2c).$

Rekurrens

Bevis (Forklaring).

- I linje 1 setter vi $H(n)$ inn i likningen.

- I linje 2 setter vi den felles faktoren r^{n-2} utenfor samtidig som vi løser opp parentesene $(n-1)$ og $(n-2)$.
- I linje 3 setter vi n utenfor de leddene hvor n er faktor.
- I linje 4 utnytter vi at r er en løsning av den karakteristiske likningen, så $ar^2 + br + c = 0$.
- For å vise at $H(n)$ er en løsning av rekurrenslikningen, er det altså nok å vise at $br + 2c = 0$
- Her skal vi bruke at r er den eneste løsningen.

Rekurrens

Bevis (Fortsatt).

- Siden vi har en 2. ordens likning, vil $a \neq 0$ og $c \neq 0$.
- Siden vi bare har en løsning, er

$$r = \frac{-b}{2a}.$$

- Setter vi dette inn i $br + 2c$ får vi

$$b \cdot \frac{-b}{2a} + 2c.$$

Rekurrens

Bevis (Fortsatt).

- Satt på felles brøkstrek er dette

$$\frac{-b^2 + 4ac}{2a}$$

som er lik 0 fordi $b^2 - 4ac = 0$.

- Det var det som gjensto å bevise, så teoremet er vist.

Rekurrens

Merk.

Vi har nå funnet to løsninger av rekurrenslikningen både i det tilfellet hvor den karakteristiske likningen har to løsninger, og i det tilfellet hvor den bare har en løsning.

Siden vi kan finne en lineær kombinasjon av slike løsninger for enhver initialbetingelse $t(1) = a$ og $t(2) = b$, har vi i realiteten funnet en generell løsning.

Hvis r og s er to forskjellige løsninger av den karakteristiske likningen, er

$$F(n) = Ar^n + Bs^n$$

den generelle løsningen av rekurrenslikningen.

Rekurrens

Merk (Fortsatt).

Hvis r er den eneste løsningen av den karakteristiske likningen er

$$F(n) = (A + Bn)r^n$$

den generelle løsningen av rekurrenslikningen.

Rekurrens

Eksempel.

- Anta at vi har fått følgende oppgave:
 - La $t(n) - t(n-1) + \frac{1}{4}t(n-2) = 0$ være en rekurrenslikning.
- a) Finn den generelle løsningen $F(n)$ til likningen.
 - b) Finn løsningen $G(n)$ som oppfyller at $G(1) = G(2) = 1$.
 - c) Finn løsningen $H(n)$ som oppfyller at $H(1) = 1$ og $H(2) = 0$.

Rekurrens

Eksempel (Fortsatt).

Først må vi finne den karakteristiske likningen

$$x^2 - x + \frac{1}{4} = 0.$$

Vi finner at $r = \frac{1}{2}$ er den eneste løsningen.

Da er den generelle løsningen

$$F(n) = (A + nB)\left(\frac{1}{2}\right)^n.$$

Dette løser a)

Rekurrens

Eksempel (Fortsatt).

For å løse punkt b), bruker vi de to initialbetingelsene til å finne to likninger med A og B som ukjente

$$- n = 1: (A + B)\frac{1}{2} = 1, \text{ det vil si } A + B = 2.$$

$$- n = 2: (A + 2B)\left(\frac{1}{2}\right)^2 = 1, \text{ det vil si } A + 2B = 4.$$

Disse har løsninger $A = 0$ og $B = 2$, så svaret på b) er

$$G(n) = 2n\left(\frac{1}{2}\right)^n.$$

Rekurrens

Eksempel (Fortsatt).

For å løse c) setter vi 1 og 0 i høyresidene i likningene over, og får

$$- A + B = 2$$

$$- A + 2B = 4$$

som har $A = 4$ og $B = -2$ som løsninger.

Løsningen på c) er derfor

$$H(n) = (4 - 2n)\left(\frac{1}{2}\right)^n.$$

Rekurrens

Spiller så rekurrenslikninger noen rolle i informatikken?

Hvis man skal analysere kompleksiteten av en algoritme, det vil si finne ut av hvor mange regneskritt som trengs som funksjon av størrelsen på input, risikerer man at resultatet blir en inhomogen rekurrenslikning. (Dette er observert på vitenskapelig foredrag om temaet).

Da vil regnetiden vokse eksponensielt med størrelsen på input.

Hvis grunntallet i denne eksponenten ikke er så mye større enn 1, er ikke det nødvendigvis ødeleggende for nytten av algoritmen.

Det er stor forskjell på en algoritme hvor regnetiden er begrenset av $1000(1,08)^n$ og en der regnetiden er begrenset av $2(1,23)^n$, hvor n er antall bit i input. n trenger ikke å være så veldig stor før den første trolig er mer effektiv enn den andre.

Rekurrens

Eksempel.

- En litt håpløs måte å sende en kryptert binær sekvens på vil være å sende 10 eller 01 valgt vilkårlig der det skulle stått 1 og 0 der det skulle stått 0.
- Det er da opp til mottageren, som er den eneste som kjenner krypteringsmåten, å liste opp alle mulige opprinnelige meldinger og finne den som gir mening.
- Hva er det maksimale antall $F(n)$ opprinnelige meldinger som kan ligge bak en mottatt bitsekvens av lengde n ?

Rekurrens

Eksempel (Fortsatt).

- Hvis $n = 1$ har vi bare en mulighet, den sendte biten er 0
- Hvis $n = 2$ har vi to muligheter, bitsekvensen representerer 1 eller bitsekvensen representerer 00.
- For $n \geq 2$ har vi to muligheter:
 - Siste siffer er 0 og representerer en 0. Det totale antall muligheter i den situasjonen er $F(n - 1)$ ettersom resten av meldingen også skal representere en bitsekvens.
 - De siste to sifrene representerer 1. Dette svarer egentlig til $F(n - 2)$ muligheter totalt.

Rekurrens

Eksempel (Fortsatt).

- Svaret på problemet får vi ved å løse rekurrenslikningen

$$t(n) = t(n - 1) + t(n - 2)$$

med initialbetingelser $t(1) = 1$ og $t(2) = 2$

- Løsningen er da at $F(n)$ er Fibonaccitall nr. $n + 1$, noe som viser at metoden er svært upraktisk.

Rekurrens

- Vi har vært lojale mot læreboka og latt løsninger av rekurrenslikninger være følger, eller funksjoner definert på \mathbb{N} .
- Det er imidlertid ikke noe i veien for at vi ser på løsninger definert på hele \mathbb{J} , eller fra 0 og oppover.
- Vi kan tolke likningen

$$t(n) - t(n - 1) - t(n - 2) = 0$$

for Fibonacci-følgen som en likning der n varierer over hele \mathbb{J} .

- Det ville eksempelvis gitt oss at vi kan finne $F(0)$ fra

$$F(2) - F(1) - F(0) = 0,$$

hvilket gir $F(0) = 0$.

Rekurrens

- Vi kan godt fortsette nedover med $F(1) - F(0) - F(-1) = 0$ og finne at $F(-1) = 1$.
- Den praktiske nytten vil være at det ofte er enklere å bestemme løsningen til en rekurrenslikning med initialverdier fra $F(0)$ og $F(1)$ fordi de lineære likningene vil bli penere.
- Hvis s og r er løsninger av den karakteristiske likningen, kan vi finne A og B fra
 - $A + B = F(0)$
 - $Ar + Bs = F(1)$
- Har vi bare en løsning r , er forenklingen ved å gå til $F(0)$ enda større.
 - $A = F(0)$
 - $(A + B)r = F(1)$
- Bruker man rekurrenslikningen til å regne ut $F(0)$ er dette en lovlig måte å løse oppgaver på.

Forelesning 17

Generell rekursjon og induksjon

Dag Normann - 10. mars 2008

Opphenting

- Forrige uke så vi på rekurrenslikninger.
- En rekurrenslikning er en funksjonslikning på formen

$$at(n) + bt(n-1) + ct(n-2) = 0$$

hvor en løsning er en funksjon

$F: \mathbb{N} \rightarrow \mathbb{R}$ slik at

$$aF(n) + bF(n-1) + cF(n-2) = 0$$

for alle $n \geq 2$ (eller for alle n hvor n , $n-1$ og $n-2$ er i definisjonsområdet til F , når vi vil bruke maskineriet vårt i en mer generell situasjon).

- Den karakteristiske likningen er da

$$ax^2 + bx + c = 0$$

Opphenting

- Hvis r og s er to forskjellige løsninger av den karakteristiske likningen, er den generelle løsningen av rekurrenslikningen

$$F(n) = Ar^n + Bs^n$$

hvor A og B er vilkårlige reelle tall.

- Hvis den karakteristiske likningen bare har en løsning r , er den generelle løsningen av rekurrenslikningen

$$(A + Bn)r^n.$$

Opphenting

- Hvis vi i tillegg har krav om at $t(1) = a$ og $t(2) = b$, kan vi bestemme A og B i den generelle løsningen ved å sette inn for $n = 1$ og $n = 2$ i den generelle løsningen, og løse likningene mhp. A og B .
- Dette vil alltid fungere.
- Boka ser bare på tilfellet med initialbetingelser på $t(1)$ og $t(2)$.
- Hadde vi satt krav til $t(17)$ og $t(256)$, eller til t -verdien for to andre, forskjellige tall, kunne vi fortsatt bestemt A og B fra den informasjonen.

- Initialbetingelser for $n = 0$ og $n = 1$ kan gi den enkleste regningen.
- Vi skal nå fortsette innføringen av nytt stoff.

Rekursjon og programmering

- Vi startet innføringen av rekursjon med å gi eksempler på hvordan vi kunne finne pseudokoder som svarer til rekursive konstruksjoner.
- Vi kan minne om at hvis
 - $f(1) = a$
 - $f(n + 1) = g(f(n), n)$

er en rekursiv funksjon, og vi har en pseudokode for g , kan vi erstatte denne pseudokoden (som en del av en større kode) med

$$x \leftarrow g(i, j)$$

i betydningen at variabelen x får verdien til f når inputvariablene får verdiene til i og j .

- Det er flere måter vi kan lage en pseudokode for g på, vi skal se på to av dem:

Rekursjon og programmering

Eksempel.

```

1 Input n [n ∈ ℕ]
2 x ← a
3 Output x
4 For i = 2 to n do
    4.1 x ← g(x, i)
    4.2 Output x

```

Merk.

Denne pseudokoden vil skrive ut $f(1), f(2), \dots, f(n)$ i rekkefølge.
Hvis vi bare vil ha ut $f(n)$ blir koden enda enklere:

Rekursjon og programmering

Eksempel.

```

1 Input n [n ∈ ℕ]

```

```

2 x ← a
3 For i = 2 to n do
    3.1 x ← g(x, i)
4 Output x

```

Rekursjon og programmering

- Læreboka har et lite avsnitt om plassen til rekursjon i de enkelte programmeringsspråk.
- Dette er lesestoff, som ikke blir utdypet på forelesningene.
- Enkelte programmeringsspråk tillater til og med en sterkere form for rekursjon, selvkallende prosedyrer.
- Rekursjon er et spesialtilfelle av selvkallende prosedyrer, hvor vi definerer en funksjon $f(x)$ ved å bruke verdier $f(y)$ for enkelte y avhengige av x .
- En slik definisjon kan lett lede til løkkeberegninger eller uendelige beregninger, uten at det er lett å se hvorfor det er tilfelle.
- Vi skal ikke komme nærmere inn på det (nå), etter følgende eksempel.

Rekursjon og programmering

Eksempel.

- Da vi ga eksempler på pseudokoder, ga vi et eksempel på en algoritme som ingen ennå vet om vil terminere for alle verdier på input, og vi ga algoritmen i form av en pseudokode.
- Våre pseudokoder fanger ikke opp muligheten for selvkallende prosedyrer, men hadde vi hatt den muligheten, kunne vi betraktet følgende som en meningsfylt algoritme:

Rekursjon og programmering

Eksempel (Fortsatt).

- La $f(1) = 1$.
- La $f(n) = f(\frac{n}{2}) + 1$ hvis n er et partall.
- La $f(n) = f(3n + 1) + 1$ hvis $n > 1$ er et oddetall.
- Vi kan da eksempelvis regne ut

$$f(20) = f(10) + 1 = f(5) + 2 = f(16) + 3$$

$$= f(8) + 4 = f(4) + 5 = f(2) + 6 = f(1) + 7 = 8$$

- f er veldefinert som en *partiell* funksjon som vi ikke vet om er *total*.

Generell induksjon og rekursjon

- Vi har argumentert for at konstruksjoner ved rekursjon virker og at induksjon er en gyldig bevisteknikk.
- Vi har begrunnet dette med at vi kan nå alle naturlige tall ved å starte med 1 og så legge til 1 så mange ganger vi trenger.
- En måte å forklare hvorfor induksjonsbevis er matematisk holdbare argumenter på er følgende:
- Vi har at \mathbb{N} er den minste mengden som oppfyller
 - $1 \in \mathbb{N}$
 - Hvis $x \in \mathbb{N}$ vil $x + 1 \in \mathbb{N}$
- Hvis vi da viser en egenskap P ved induksjon, viser vi at
 - $1 \in \{y : P(y)\}$
 - Hvis $x \in \{y : P(y)\}$ vil $x + 1 \in \{y : P(y)\}$.
- Siden \mathbb{N} var den minste mengden med denne egenskapen, må $\mathbb{N} \subseteq \{y : P(y)\}$, eller, som vi sier, $P(x)$ holder for alle $x \in \mathbb{N}$.

Generell induksjon og rekursjon

- I logikk og informatikk (og også innen andre deler av matematikken og i andre fag) ser man definisjoner som likner på vår beskrivelse av \mathbb{N} ; man beskriver en strukturert mengde ved å si hva som kommer inn som start og hvordan man finner mer komplekse elementer av mengden.
- Innen informatikk og logikk beskriver vi gjerne formelle språk på den måten.
- Det er et uttrykt ønske fra informatikerne om at de studentene som skal studere videre hos dem, skal ha en bedre forståelse av induksjon og rekursjon enn hva som har vært tilfelle tidligere år, og at studentene skal ha kjennskap til rekursjon over andre strukturer enn \mathbb{N} .

Generell induksjon og rekursjon

- I de følgende eksemplene skal vi anta at vi har et alfabet som består av alle de symbolene vi kan finne på tastaturet til en standard datamaskin (norsk standard om presisjonen er nødvendig), hvor tomrom er å betrakte som et eget symbol. Vi bruker bokstaver i *kursiv* som variable over bokstavene på tastaturet, og vi kan bruke andre bokstaver i kursiv som variable for ord, hvor:

Generell induksjon og rekursjon

- Et ord er en ordnet sekvens av bokstaver, hvor bokstavene er skrevet uten ekstra tegn i mellom.
- Det er vanlig å la e betegne det tomme ordet.
- Vi føyer en bokstav til et ord ved ganske enkelt å skrive det inntil ordet på høyre side.
- Dette kan vi bruke til å gi en induktiv beskrivelse av mengden av ord.

Generell induksjon og rekursjon

Definisjon.

Mengden av ord er den minste mengden som oppfyller

- Det tomme ordet ϵ er et ord.
- Hvis w er et ord og b er en bokstav, er wb et ord.

Generell induksjon og rekursjon

Merk.

- Dette eksemplet virker en smule kunstig, fordi vi ikke oppfatter ord slik, selv om de fleste av oss starter med tom linje, og så skriver en og en bokstav fra venstre mot høyre.
- Det spiller imidlertid en rolle for hvordan ord representeres som data om de sees på som ordnede sekvenser med en gitt lengde eller som bygget opp ved at nye bokstaver legges til.
- Ord, slik vi har definert dem, er et spesialtilfelle av lister.
- En liste oppfattes som oftest som at enten er den den tomme listen ϵ , eller så er den et ordnet par av siste element og resten av listen.
- Mange tenker seg at ytterste element (*hodet*) i en liste står til venstre for resten (*halen*) av listen.
- Programmeringspråket LISP er basert på listerekursjon.

Generell induksjon og rekursjon

Eksempel.

Som et eksempel på en rekursivt definisjon av en funksjon på mengden av ord kan vi betrakte PL (Push Left) som virker på et ord w og en bokstav a ved:

- $PL(\epsilon, a) = a$
- $PL(wb, a) = PL(w, a)b$

Det som her skjer er at PL tar for seg et ord w og en bokstav a , og skriver bokstaven *foran* ordet, slik at vi får aw .

Egentlig burde vi vist denne egenskapen ved PL ved induksjon, men vi avstår i denne omgangen.

Generell induksjon og rekursjon

Eksempel (Fortsatt).

Følgende regne-eksempel viser hvordan PL virker:

$$PL(aba, c) =$$

$$PL(ab, c)a =$$

$$PL(a, c)ba =$$

$$PL(e, c)aba =$$

$$caba$$

Generell induksjon og rekursjon

Eksempel.

Med utgangspunkt i eksemplet PL fra forrige side skal vi definere en speilingsfunksjon R ved rekursjon. R vil ta et ord som input og output vil være ordet skrevet baklengs:

- Vi definerer R ved:
 - $R(e) = e$
 - $R(wb) = PL(R(w), b)$

Igjen overlater vi bekreftelsen av at funksjonen virker i henhold til spesifikasjonen til den enkelte.

Vi skal se på et eksempel, og i tillegg gi en oppgave, som skal hjelpe til med dette.

Generell induksjon og rekursjon

Eksempel.

Vi skal vise ved et regneeksempel i full detalj at $R(abc) = cba$ slik R og PL er definerte.

Vi vil bare bruke symbolet e når vi ellers måtte skrevet *ingenting*, så eab er det samme som ab .

$$\begin{aligned} R(abc) &= PL(R(ab), c) \\ &= PL(PL(R(a), b), c) \\ &= PL(PL(PL(R(e), a), b), c) \\ &= PL(PL(PL(e, a), b), c) = PL(PL(a, b), c) \\ &= PL(PL(e, b)a, c) = PL(ba, c) \\ &= PL(b, c)a = PL(e, c)ba = cba. \end{aligned}$$

Generell induksjon og rekursjon

Oppgave.

a) Ved å bruke den rekursive definisjonen av PL, vis hvordan vi skritt for skritt kan finne verdiene av

- PL(e, d)
- PL(a, d)
- PL(ab, d)
- PL(aba, d)

Husk at *variablene* a og b kan stå for hvilken som helst av *bokstavene* a, b og d.

b) Vis, ved å bruke definisjonen av R og egenskapen til PL at $R(abac) = caba$.

Generell induksjon og rekursjon

Eksempel.

Den siste funksjonen vi skal se på i forbindelse med den induktive definisjonen av mengden av ord er sammensetningsfunksjonen $w * v = wv$.

Denne kan også defineres ved rekursjon som følger:

- $w * e = w$
- $w * (vb) = (w * v)b$

Vi kunne gjort det vanskeligere, nemlig brukt rekursjon på første argument:

- $S(e, v) = v$
- $S(wb, v) = S(w, PL(v, b))$

Hvis vi nå vil vise at $S(w, v) = w * v$ for alle ord w og v , kan vi ha god bruk for induksjon.

Generell induksjon og rekursjon

- Automatateori er den matematiske teorien for studiet av formelle regnemaskiner som tar for seg et inputord, og enten prosesserer et outputord eller svarer på et spørsmål om inputordet.
- Flere av disse formelle maskinene leser inputordet fra venstre mot høyre, og utfører en beregning underveis.
- Eksempler på dette er de såkalte endelige tilstandsmaskinene og pushdownautomater eller stackautomater.
- Hvordan en slik automat virker på et ord defineres ved rekursjon på oppbyggingen av ordet.
- Vi skal ikke innføre automatateori, men vi skal i eksempler og oppgaver se på et par tilfeller hvor rekursjon på ord kan brukes til å erstatte slike formelle regnemaskiner.
- De kraftigste formelle maskinene fanges ikke opp av ord-rekursjon.

Generell induksjon og rekursjon

Eksempel.

- La w være et ord hvor vi vet at bare bokstavene a og b forekommer.
- Vi vil finne en algoritme som avgjør om det er like mange bokstaver av hvert slag, eller om det er et overskudd av den ene.
- Som en hjelpefunksjon definerer vi en funksjon $f(w)$ slik at $f(w)$ er differansen mellom antall a 'er og antall b 'er i w .

$$- f(e) = 0$$

$$- f(wa) = f(w) + 1$$

$$- f(wb) = f(w) - 1$$

Generell induksjon og rekursjon

Eksempel (Fortsatt).

- Vi kan da regne ut

$$f(aababba) = f(aababb) + 1 = f(aabab) - 1 + 1 = f(aabab)$$

$$= f(aaba) - 1 = f(aab) = f(aa) - 1 = f(a) = f(e) + 1 = 1.$$

Generell induksjon og rekursjon

Eksempel (Fortsatt).

- Siden ord er ordnede sekvenser av symboler, kan vi finne pseudokoder som erstatter ord-rekursjon.
- Vi skal gi en pseudokode for beregning av f fra dette eksemplet.
- Her er det viktig at *input* er et ord hvor bokstavene a og b forekommer, og at *output* er et helt tall.
- Vanligvis må man deklare typene til variablene som en del av programmet, men det formaliserer vi ikke her.

Generell induksjon og rekursjon

Eksempel (Fortsatt).

1 *Input* $n [n \in \mathbb{N}_0]$

```

2 Input  $w$  [ $w = v_1 \cdots v_n$  er et ord av lengde  $n$ ]
3  $x \leftarrow 0$ 
4 For  $i = 1$  to  $n$  do
    4.1 If  $v_i = a$  then
        4.1.1  $x \leftarrow x + 1$ 
        else
        4.1.2  $x \leftarrow x - 1$ 
5 Output  $x$ 

```

Generell induksjon og rekursjon

Merk.

- Hvis vi arbeider med algoritmer som skal virke på ord i et alfabet eller lister av data, vil det være aktuelt å innføre egne kontrollstrukturer for listerekursjon.
- Det finnes programmeringsspråk av teoretisk interesse, og også av praktisk betydning, hvor det er enkelt å uttrykke listerekursjon og andre former for generell rekursjon.

Generell induksjon og rekursjon

Fra nå av skal vi anta at vi har definert mengden av ord, og at vi kan foreta oss de operasjoner på ord og bokstaver som vi finner nødvendige.

Hvorvidt det er naturlig å bruke *ordrekursjon* eller ikke, avhenger av hvordan vi representerer ord i en datamaskin.

Definisjon.

Et formelt språk er en mengde av ord.

Generell induksjon og rekursjon

Eksempel.

- Mengden av syntaktisk korrekte program i et gitt programmeringsspråk er et formelt språk.
- Mengden av utsagnslogiske formler i utsagnsvariablene p , q og r er et formelt språk, hvis vi regner \neg , \wedge , \vee , \rightarrow og \leftrightarrow med blant bokstavene.

- Mengden av ord fra *delalfabetet* $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ som uttrykker kubikktall er et formelt språk.
- Mengden av ord som betyr noe både på tysk og spansk er et formelt språk.

Generell induksjon og rekursjon

Merk.

- Mange av de språkene som oppstår i en informatikk-sammenheng er definert induktivt.
- Den begrensningen vi ga til at symbolene skal finnes på tastaturet, er ikke vanlig.
- Det vanlige er at man for ethvert formelt språk også presiserer hvilket alfabet som skal brukes.
- Vår opprinnelige definisjon av formelle språk er brukbar for studiet av programmeringsspråk, men det er viktige eksempler som går ut over begrensningen til tastaturet.

Generell induksjon og rekursjon

Eksempel.

- Vi skal nå se hvordan vi kan gi en helt presis definisjon av de utsagnslogiske formlene.
- For at ikke eksemplet skal bli for omfattende, dropper vi bindeordene \rightarrow og \leftrightarrow i denne definisjonen.
- Vi lar p , q og r være utvalgte bokstaver.
- Vi skal gi en induktiv definisjon av mengden av utsagnslogiske formler hvor vi bruker p , q og r som utsagnsvariable, bindeordene \neg , \wedge og \vee , og hvor vi har formalisert bruken av parenteser.
- Hva vi mener med *induktiv* definisjon vil vi la komme frem gjennom eksemplene.

Generell induksjon og rekursjon

Eksempel (Fortsatt).

1. p , q og r er formler.
2. Hvis A er en formel, er $\neg A$ en formel.
3. Hvis A og B er formler, er $(A \wedge B)$ og $(A \vee B)$ formler.
4. Mengden av formler er den minste mengden som oppfyller 1. - 3. over.

- Når man blir vant til å arbeide med induktive definisjoner, vil vanligvis punktet tilsvarende 4. være underforstått.

- Definisjonen av *ord* hadde langt på vei det samme formatet.

Rekursjon og programmering

Eksempel (Fortsatt).

- Vi har allerede gitt en viktig definisjon ved rekursjon på oppbyggingen av en formel, uten at vi har sagt direkte at det var det vi gjorde.
- En sannhetsverdifunksjon er en funksjon hvor definisjonsområdet er mengden av fordelinger av sannhetsverdier til variablene p , q og r , og verdiområdet er $\{T, F\}$.
- Ved hjelp av sannhetsverditabeller har vi definert funksjonene F_{\neg} , F_{\wedge} og F_{\vee} , hvor eksempelvis $F_{\neg}(F) = T$, $F_{\wedge}(T, F) = F$ og $F_{\vee}(T, F) = T$.
- Hvis (s_p, s_q, s_r) er en ordnet sekvens av tre sannhetsverdier, definerer vi

$$F_A(s_p, s_q, s_r)$$

ved rekursjon på oppbyggingen av A som følger:

Generell induksjon og rekursjon

Eksempel (Fortsatt).

- $F_p(s_p, s_q, s_r) = s_p$
- $F_q(s_p, s_q, s_r) = s_q$
- $F_r(s_p, s_q, s_r) = s_r$
- $F_{\neg A}(s_p, s_q, s_r) = F_{\neg}(F_A(s_p, s_q, s_r))$
- $F_{(A \vee B)}(s_p, s_q, s_r) = F_{\vee}(F_A(s_p, s_q, s_r), F_B(s_p, s_q, s_r))$
- $F_{(A \wedge B)}(s_p, s_q, s_r) = F_{\wedge}(F_A(s_p, s_q, s_r), F_B(s_p, s_q, s_r))$
- For å kunne forstå denne definisjonen trenger vi mye av det vi har lært, blant annet:
 - Generelle funksjoner
 - Sammensetning av funksjoner
 - Rekursive definisjoner i en generell situasjon.

Generell induksjon og rekursjon

Eksempel (Fortsatt).

- Vi bestemmer hvordan sannhetsverdifunksjonen skal se ut direkte for de enkleste formelene, nemlig utsagnsvariablene.
- For sammensatte formler vil sannhetsverdifunksjonen avhenge av hvilke sannhetsverdifunksjoner vi har for delformler.

- Det er akkurat denne delen av rekursjonen vi følger når vi skriver ut en sannhetsverdi-tabell.
- Hvis vi skal bruke utsagnslogikk i en programmeringssammenheng, vil det være programmer basert på en slik rekursiv definisjon som ligger til grunn når maskinen beregner verdien av en Boolsk test.

Forelesning 18

Generell rekursjon og induksjon

Dag Normann - 12. mars 2008

Generell induksjon og rekursjon

- Mandag så vi på induktivt definerte mengder og noen eksempler på funksjoner definert ved rekursjon over oppbyggingen av elementene i en slik induktivt definert mengde.
- Vi så på mengden av ord over et alfabet, og på hvordan visse operasjoner på ord kunne defineres ved hjelp av rekursjon.
- Avslutningsvis definerte vi mengden av utsagnslogiske formler hvor vi begrenser oss til utsagnsvariablene p , q og r , og til bindeordene \neg , \wedge og \vee .
- Vi så hvordan utarbeidelsen av sannhetsverditabellen til en formel kan oppfattes som en rekursiv prosess, vi bygger den opp nedenfra ved å gå til mer og mer komplekse formler.

Generell induksjon og rekursjon

- I vårt neste eksempel skal vi se på en definisjon hvor vi bruker simultan rekursjon.
- Simultan rekursjon innebærer at vi definerer to funksjoner f og g samtidig ved rekursjon.
- For å illustrere hva vi mener med "samtidig" skal vi se på et enkelt eksempel:

Generell induksjon og rekursjon

Eksempel.

- La $f(1) = 1$
- La $g(1) = 2$
- La $f(n + 1) = f(n) \cdot g(n)$ for alle $n \in \mathbb{N}$.
- La $g(n + 1) = f(n) + g(n)$ for alle $n \in \mathbb{N}$.
- Vi ser at da kan vi regne ut $f(2) = 1 \cdot 2$ og $g(2) = 1 + 2 = 3$.
- Da er $f(3) = 2 \cdot 3 = 6$ mens $g(3) = 2 + 3 = 5$.
- Slik kan vi fortsette å regne ut resultatene parvis, men vi kommer ingen vei hvis vi bare prøver å regne ut verdiene til f eller bare verdiene til g , vi må regne ut begge funksjonene *simultant*, det vil si *samtidig*.
- Begrunnelsen for at simultan rekursjon er en lovlig definisjonsform er den samme som for vanlig rekursjon.

Generell induksjon og rekursjon

Eksempel (Fortsatt).

- Hvis vi lar $h(n) = (f(n), g(n))$, det vil si, det ordnede paret av $f(n)$ og $g(n)$, er h en funksjon definert på \mathbb{N} og med verdiområde $\mathbb{N} \times \mathbb{N}$.
- Lar vi $t : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$ være definert ved

$$t(a, b) = (a \cdot b, a + b)$$

ser vi at vi kan erstatte definisjonen av f og g med følgende rekursive definisjon av h :

- $h(1) = (1, 2)$
- $h(n + 1) = t(h(n))$

- Det er ikke noe spesielt med dette eksemplet, vi kunne gjort en tilsvarende omskrivning hver gang vi definerer funksjoner ved simultan rekursjon.

Generell induksjon og rekursjon

Eksempel.

Det er ofte mye lettere å studere egenskapene til en utsagnslogisk formel, eksempelvis å undersøke om den er en tautologi, en kontradiksjon eller noe annet, om negasjonstegnet bare står i posisjon rett foran en utsagnsvariabel.

En slik formel sies å være på svak normalform.

Intuitivt kan vi finne en svak normalform ved å bruke deMorgans lover til å skyve negasjonstegnet innover.

Skal vi formulere dette presist, må vi bruke formatet til rekursive definisjoner.

Vi må definere den svake normalformen $SF(A)$ til A og den svake normalformen $SF_{-}(A)$ til $\neg A$ simultant.

Generell induksjon og rekursjon

Eksempel (Fortsatt).

- La $SF(A) = A$ og $SF_{-}(A) = \neg A$ hvis A er en utsagnsvariabel p , q eller r .
- $SF(\neg A) = SF_{-}(A)$
- $SF_{-}(\neg A) = SF(A)$
- $SF(A \vee B) = SF(A) \vee SF(B)$, $SF(A \wedge B) = SF(A) \wedge SF(B)$.
- $SF_{-}(A \vee B) = SF_{-}(A) \wedge SF_{-}(B)$, $SF_{-}(A \wedge B) = SF_{-}(A) \vee SF_{-}(B)$

- Heller ikke dette eksemplet er valgt bare for å gjøre MAT1030 vanskelig.

- Når man skal skrive programmer for å undersøke om en utsagnslogisk formel er oppfylld eller en kontradiksjon, er det til stor hjelp om man kan anta at formelen er på svak normalform.

Generell induksjon og rekursjon

Eksempel.

Som et eksempel skal vi se på hvordan vi finner den svake normalformen til $(\neg p \wedge q) \vee \neg(p \wedge \neg r)$:

$$\begin{aligned} SF((\neg p \wedge q) \vee \neg(p \wedge \neg r)) &= \\ SF(\neg p \wedge q) \vee SF(\neg(p \wedge \neg r)) &= \\ (SF(\neg p) \wedge SF(q)) \vee SF_{\neg}(p \wedge \neg r) &= \\ (SF_{\neg}(p) \wedge q) \vee (SF_{\neg}(p) \vee SF_{\neg}(\neg r)) &= \\ (\neg p \wedge q) \vee (\neg p \vee SF(r)) &= \\ (\neg p \wedge q) \vee (\neg p \vee r). & \end{aligned}$$

Generell induksjon og rekursjon

Eksempel.

- Riktig bruk av parenteser er viktig for at programmer skal være syntaktisk korrekte, eller for at matematiske uttrykk skal gi mening.
- I dette eksemplet skal vi se på mengden av korrekte parentesuttrykk, hvor vi bare bruker "(" og ")" .
- Vi skal se på hvordan vi kan bygge opp mengden av korrekte parentesuttrykk ved induksjon.
- Deretter skal vi vise at den samme mengden kan beskrives på en annen måte, en måte vi kan bruke for maskinell kontroll av om parenteser er satt riktig eller ikke.
- Vi trenger induksjonsbevis både over oppbyggingen av korrekte parentesuttrykk og over de ikke-negative hele tallene for å vise dette.

Generell induksjon og rekursjon

Eksempel (Fortsatt).

- Til sist skal vi vise hvordan den alternative skrivemåten kan brukes til å finne en pseudokode som avgjør om et uttrykk med parenteser er korrekt eller ikke.
De korrekte parentesuttrykkene er den minste mengden av ord som oppfyller
1. Det tomme ordet ϵ er et korrekt parentesuttrykk.

2. Hvis v er et korrekt parentesuttrykk, er $w = (v)$ et korrekt parentesuttrykk.
3. Hvis u og v er korrekte parentesuttrykk, er *sammensetningen* $w = uv$ et korrekt parentesuttrykk.

Vi skal studere dette viktige formelle språket litt nærmere.

Generell induksjon og rekursjon

Eksempel (Fortsatt).

Påstand 1

Hvis w er et korrekt parentesuttrykk, vil antall høyre og venstreparenteser i w være det samme.

Bevis

Vi bruker induksjon på oppbyggingen av et parentesuttrykk.

I dette tilfellet er påstanden egentlig opplagt, men vi beviser den ved induksjon likevel, som et eksempel.

Hvis $w = e$ er antall høyre og venstreparenteser begge lik 0, og derfor like.

La $w = (v)$ og anta at påstanden holder for v .

Generell induksjon og rekursjon

Eksempel (Fortsatt).

Da er både antall venstre- og høyreparenteser i w en større enn tilsvarende tall for v , som er like.

Derfor er de like for w også.

La så $w = uv$ og anta at påstanden holder for både u og for v .

Antall venstreparenteser i w er summen av antall venstreparenteser i u og antall venstreparenteser i v .

Ved induksjonsantagelsen er dette det samme som antall høyreparenteser i u og i v .

Da må summen også være den samme.

Generell induksjon og rekursjon

Eksempel (Fortsatt).

- Vi definerte mengden av de korrekte parentesuttrykkene som den minste mengden X slik at

- $e \in X$
- $v \in X \Rightarrow (v) \in X$
- $u \in X \wedge v \in X \Rightarrow uv \in X$
- Vi har bevist at mengden Y av ord som har like mange venstre- og høyreparenteser, og ingen andre symboler, oppfyller
 - $e \in Y$
 - $v \in Y \Rightarrow (v) \in Y$
 - $u \in Y \wedge v \in Y \Rightarrow uv \in Y$
- Det betyr at $X \subseteq Y$, og det er en reformulering av Påstand 1.

Generell induksjon og rekursjon

Eksempel (Fortsatt).

Påstand 2

Hvis w er et korrekt parentesuttrykk, og vi leser w fra venstre mot høyre, finner vi ikke noe sted hvor det har vært flere høyre- enn venstreparenteser.

Bevis

Igjen bruker vi induksjon på oppbyggingen av w som et korrekt parentesuttrykk.

Hvis $w = e$ er dette opplagt riktig.

Anta at påstanden holder for v , og la $w = (v)$.

Når vi leser w fra venstre mot høyre har vi alltid en høyreparentes mer når vi leser w enn når vi er på tilsvarende sted i v .

Generell induksjon og rekursjon

Eksempel (Fortsatt).

Det betyr at vi har et positivt overskudd av (hele tiden mens vi leser v -delen av (v) , og vi får ikke noe overskudd av) til slutt heller.

Anta at påstanden holder for u og for v , og at $w = uv$.

Når vi leser w fra venstre mot høyre kan vi først ikke opparbeide noe overskudd av) mens vi leser u -delen, og deretter heller ikke mens vi leser v -delen.

Dermed holder påstanden for w også.

Generell induksjon og rekursjon

Eksempel (Fortsatt).

Påstand 3

La w være et ord hvor vi bare har brukt symbolene "(" og ")".

Hvis w oppfyller konklusjonene i Påstand 1 og Påstand 2, vil w være et korrekt parentesuttrykk.

Bevis

Her lønner det seg å bruke induksjon på lengden av ordet w .

I motsetning til vanlig induksjon, starter beviset her med lengde 0, som er lengden til det tomme ordet.

Hvis lengden av w er 0, det vil si hvis $w = e$, er w et korrekt parentesuttrykk pr. definisjon.

La $w \neq e$ og anta at påstanden holder for alle kortere ord v (Kommentar kommer senere).

Generell induksjon og rekursjon

Eksempel (Fortsatt).

Vi må dele beviset opp i to tilfeller:

Tilfelle 1: Når vi leser w fra venstre mot høyre finner vi ikke noe sted før til slutt at w har like mange høyre- som venstreparenteser.

Siden w oppfyller påstandene, må w være på formen (v) , og siden vi er i tilfelle 1 vil også v oppfylle Påstand 1 og Påstand 2.

Ved induksjonsantagelsen er v korrekt, og da er $w = (v)$ korrekt.

Generell induksjon og rekursjon

Eksempel (Fortsatt).

Tilfelle 2: Vi kan dele w opp i to ekte delord, $w = uv$, slik at u har like mange venstre- som høyreparenteser.

Da må det samme gjelde for v , siden det gjelder for w .

Videre får vi ikke noe overskudd av høyreparenteser mens vi leser u , siden det er det samme som å lese w . Siden vi ikke starter med noe overskudd av venstreparenteser etter å ha lest u , vil situasjonen være lik om vi leser v direkte eller etter å ha lest u .

Det betyr at Påstand 1 og Påstand 2 holder både u og v .

Generell induksjon og rekursjon

Eksempel (Fortsatt).

Ved induksjonsantagelsen er da u og v korrekte.

Da er $w = uv$ også korrekt.

Siden dette argumentet dekker alle mulighetene, er Påstand 3 vist ved induksjon over \mathbb{N}_0

Generell induksjon og rekursjon

Eksempel (Fortsatt).

- La oss oppsummere dette eksemplet.
- For å vise Påstand 1 brukte vi induksjon over oppbyggingen av et uttrykk som et korrekt parentesuttrykk.
- Vi brukte induksjon over den samme induktivt definerte mengden for å bevise Påstand 2.
- For å bevise Påstand 3 brukte vi imidlertid induksjon over \mathbb{N}_0 .
- Det er mulig å skrive om bevisene for Påstand 1 og for Påstand 2 slik at de blir beviser ved induksjon over \mathbb{N}_0 , eksempelvis bruke induksjon på antall parenteser i uttrykket.
- Dette er en omskrivning som i innlæringsfasen kan gjøre det lettere å forstå generell induksjon, men som på sikt gjør det mer tungvint å formulere bevisene.

Generell induksjon og rekursjon

Eksempel (Fortsatt).

- Noen vil stusse over at vi ikke brukte den vanlige formuleringen med å vise $P(1)$ og at $P(n) \Rightarrow P(n+1)$ for alle n .
- Det vi brukte her var et annet induksjonsprinsipp:
Anta at vi kan vise for alle n at

$$\forall m < n P(m) \rightarrow P(n)$$

- Da kan det ikke finnes noe minste tall n slik at $P(n)$ er usann.
- Dermed vet vi at $P(n)$ er sann for alle n .
- Det er fullt ut akseptabelt å bruke denne alternative formen i bevis.
- Vi skal komme tilbake med et eksempel til på denne formen for induksjon.

Generell induksjon og rekursjon

Eksempel (Fortsatt).

- Hvordan kan vi så finne en pseudokode som avgjør om et parentesuttrykk er korrekt eller ikke?
- Vi skriver en kode for funksjonen som teller overskudd av "(" i forhold til ")".
- Underveis kontrolleres det at vi ikke får for mange ")".
- Til sist kontrolleres det at vi hadde like mange av hver.
- n hentes fra \mathbb{N}_0 , alle v_i 'ene er parenteser, x varierer over \mathbb{J} og y tar JA/NEI som mulige verdier.
- Pseudokoden kommer på neste side.

Generell induksjon og rekursjon

```
1 Input n
2 Input w = v1 ··· vn
3 x ← 0
4 y ← JA
5 For i = 1 to n do
    5.1 If vi = ( then
        5.1.1 x ← x + 1
        else
        5.1.2 x ← x - 1
    5.2 If x < 0 then
        5.2.1 y ← NEI
6 If x > 0 then
    6.1 y ← NEI
7 Output y
```

Generell induksjon og rekursjon

Merk.

- I virkelighetens verden har vi flere typer parenteser å holde styr på.
- Skal vi lage en algoritme som virker i en slik situasjon, kan vi ikke la x ta verdier i mengden av hele tall, men i mengden av lister av venstreparenteser.
- Hver gang vi treffer på en venstreparentes av et slag, legger vi den til listen.
- Hver gang vi treffer på en høyreparentes, må vi sammenlikne den med venstreparentesen ytterst i listen (hodet).

- Vi trenger et språk for håndtering av lister for å kunne skrive pseudokoder basert på denne skissen.
- For de som senere lærer om “push-down”-automater, vil det å lage en automat som realiserer denne algoritmen være en enkel oppgave.

Generell induksjon og rekursjon

Eksempel.

- La oss gå tilbake til den formen for induksjon som vi brukte for å vise at alle parentesuttrykk som oppfyller konklusjonene i Påstand 1 og 2 vil være korrekte.
- Da vi diskuterte kontrapositive argumenter, ga vi et eksempel på hvordan man kan vise at alle tall $n \geq 2$ kan faktoriseres i primtall (hvor det å konstatere at et tall er et primtall betraktes som en faktorisering).
- Det er lettere å formulere beviset hvis man kan bruke induksjon.
- La $n \geq 2$, og anta at påstanden holder for alle m slik at $2 \leq m < n$.
- Hvis n ikke er et primtall, finnes tall $m < n$ og $k < n$ slik at $n = mk$.

Generell induksjon og rekursjon

Eksempel (Fortsatt).

- Ved antagelsen kan både m og k faktoriseres i primtall.
- Ved å bruke alle disse primtallsfaktorene sammen, får vi en primtallsfaktorisering av n

Merk.

Dette argumentet gir oss ikke lov til å konkludere at det finnes nøyaktig en måte å faktorisere et tall på, bare at det finnes minst en.

Generell induksjon og rekursjon

- I eksemplet over viste vi en egenskap $P(n)$ ut fra antagelsen om at $P(m)$ og $P(k)$ holder for to utvalgte tall $m < n$ og $k < n$.
- Vi konkluderte med at $P(n)$ må holde for alle n .
- Vi kan gi en generell kontrapositiv begrunnelse for at det må være slik.
- Anta at det finnes et tall n_1 slik at $P(n_1)$ ikke holder.
- Ved argumentet vårt må det da finnes $n_2 < n_1$ slik at heller ikke $P(n_2)$ holder.

- Vi kan fortsette med å finne $n_3 < n_2$, $n_4 < n_3$ osv. slik at $P(n_i)$ ikke holder for noen $i \in \mathbb{N}$.
- *!* Det finnes imidlertid ingen strengt synkende følge av naturlige tall, så dette er umulig.
- Vi kan i prinsippet bruke induksjon som bevisform for enhver ordning som ikke tillater noen uendelig strengt synkende følge.
- Vi skal ikke presse denne sitronen lenger, det er sikkert surt nok for mange.

Generell induksjon og rekursjon

- Eksempelene med rekursjon på oppbyggingen av ord eller formler er så sentrale at de kan bli brukt som grunnlag for oppgaver gitt til eksamen eller til Oblig. 2.
- Det neste eksemplet har ikke den samme betydningen i informatikk, og kan derfor best sees på som et eksempel for innøvelse av forståelse og ferdigheter.
- I Oblig. 1 så vi på hvordan vi kunne beskrive ordnede par innenfor rammen av mengdelære.
- I realiteten kan alle data det måtte være aktuelt å gi en digital representasjon for kunne beskrives innenfor rammen av mengdelæren.
- I eksemplet beskriver vi “den universelle datatype”, en universell mengde hvor vi kan finne alle *virkelige* datatyper som delmengder.

Generell induksjon og rekursjon

Eksempel.

- Vi definerer de hereditært endelige mengdene HF som den minste mengden slik at
 - $\emptyset \in \text{HF}$
 - Hvis $X = \{a_1, \dots, a_n\}$ er en endelig delmengde av HF, så er $X \in \text{HF}$.
- Vi definerer $f : \text{HF} \rightarrow \mathbb{N}_0$ ved rekursjon over HF ved
 - $f(\emptyset) = 0$
 - $f(\{a_1, \dots, a_n\}) = 2^{f(a_1)} + \dots + 2^{f(a_n)}$.

Her må vi anta at mengden er beskrevet uten gjentakelse, det vil si at alle a_i 'ene er forskjellige.

Generell induksjon og rekursjon

Eksempel (Fortsatt).

- Ved induksjon over HF ser vi at $f(X) \in \mathbb{N}_0$ for alle $X \in \text{HF}$.
Hvis $X = \emptyset$ ser vi det direkte, og hvis $X = \{a_1, \dots, a_n\}$ kan vi bruke induksjonsantagelsen som sier at $f(a_i) \in \mathbb{N}_0$ for alle i .
- Hvis $k \in \mathbb{N}_0$ vil det finnes en og bare en $X \in \text{HF}$ slik at $f(X) = k$.

- Her bruker vi induksjon på k .

For $n = 0$ ser vi at det bare er $X = \emptyset$ som kan gi $f(X) = 0$ siden $2^{f(a_i)} > 0$ for alle mulige $a_i \in X$.

For $k > 0$, finnes det en og bare en måte å skrive k som en sum av forskjellige 2^r -erpotenser på, og hver eksponent k_i vil komme fra en og bare en mengde a_i .

Det betyr at vi kan finne X slik at $f(X) = k$ fra k .

Generell induksjon og rekursjon

Oppgave.

La HF og f være definert som i eksemplet over, og la $S : HF \rightarrow HF$ være definert ved

$$S(X) = X \cup \{X\}.$$

- Vis at $X \in HF \Rightarrow S(X) \in HF$.
- La $X, Y \in HF$.
Vis at $X \in Y \Rightarrow f(X) < f(Y)$.
- Forklar hvorfor det ikke finnes noen $X \in HF$ slik at $X \in X$.
- La $N \subseteq HF$ være den minste mengden slik at $\emptyset \in N$ og slik at hvis $X \in N$ så vil $S(X) \in N$.
Drøft sammenhengen mellom N og \mathbb{N}_0 .

Generell induksjon og rekursjon

- Hvis vår induktive definisjon av mengden av utsagnslogiske formler skulle vært gitt i en lærebok i informatikk på et avansert nivå, eller i en forskningsartikkel, ville den sett ut som noe slikt:

Utsagnsvariabel v

$v ::= |p|q|r$

Formel A

$A ::= v | \neg A | (A \wedge A) | (A \vee A)$

- De to første linjene skal leses som følger:

I denne definisjonen lar vi v betegne en vilkårlig utsagnsvariabel.

v kan stå for p , q eller r .

- De to neste linjene skal leses som følger:

I denne definisjonen lar vi A betegne en vilkårlig formel.

En formel kan enten være en utsagnsvariabel, fremkommet fra en annen formel ved å skrive \neg foran, eller fremkommet fra to andre formler ved å skrive \wedge eller \vee mellom, og parenteser rundt.

Generell induksjon og rekursjon

- Vi kan bruke den samme effektive notasjonen for å definere mengden av korrekte parentesuttrykk:

Parentesuttrykk P

$P ::= e|(P)|PP$

som uttrykker at vi får parentesuttrykkene ved å starte med det tomme ordet e og deretter enten sette parenteser rundt et korrekt uttrykk eller sette to korrekte uttrykk sammen.

- Som tidligere nevnt, liker informatikere og logikere å la de naturlige tallene starte med 0, altså å arbeide med \mathbb{N}_0 i stedet for \mathbb{N} .
- Følgende definisjon er påtruffet i informatikk-literatur:

$n : \text{NAT}$

$n ::= 0|S(n).$

- Dette skal leses som at vi definerer en datatype NAT ved å la n stå for et vilkårlig objekt, og vi finner objektene i NAT, enten som *symbolet* 0 eller som en *symbolsekvens* $S(w)$ hvor w er en symbolsekvens vi allerede vet er av type NAT.

Programmet videre

- Plenumsregningen 13.03 er avlyst av personlige grunner.
- Ingen undervisning uke 12 grunnet Påske.
- Ingen forelesning 24.03 av samme grunn.
- Plenumsregningen 27.03 avlyses ikke likevel, men den kan bli flyttet.
- Uke 14, 31.3 - 4.4, er det ingen undervisning grunnet eksamensavvikling i andre emner.
- Uke 15 tar Roger Antonsen all undervisning.

GOD PÅSKE

Forelesning 19

Kombinatorikk

Dag Normann - 26. mars 2008

Oppsummering

- Før påske gikk vi gjennom kapitlene 1-7 i læreboka.
- De omfattet
 - Eksempler på algoritmer og bruk av pseudokoder.
 - Forskjellige tallsystemer.
 - Hvordan regning med hele tall og reelle tall foregår inne i en datamaskin.
 - Bruk av utsagnslogiske bindeord og kvantorer.
 - Mengder, relasjoner og funksjoner.
 - Induksjon og rekursjon.

Oppsummering

- Under forelesningene om relasjoner og funksjoner innførte vi mange nye begreper på kort tid.
- Vi beskrev hva det vil si at en relasjon er
 - Symmetrisk.
 - Antisymmetrisk.
 - Refleksiv.
 - Irrefleksiv.
 - Transitiv.
- Dette er begreper man bør kjenne til.

Oppsummering

- Under gjennomgangen av funksjoner lastet vi også på med mange nye begreper:
 - Definisjonsområde.
 - Verdiområde.
 - Bildemengde.
 - Injektiv funksjon.
 - Surjektiv funksjon.
 - Sammensetning av funksjoner.
 - Inverse funksjoner.
- Dette bør dere også kjenne til.

Oppsummering

- Vi fulgte boka da vi gikk igjennom
 - vanlige rekursive definisjoner.
 - bevis ved induksjon.
 - fremgangsmåte for løsning av rekurrenslikninger.
- Dette er kjernestoff, og meget sentralt i pensum.
- I tillegg så vi på andre induktivt definerte strukturer som
 - mengden av ord over et alfabet.
 - mengden av utsagnslogiske formler.
 - mengden av parentesuttrykk.

Oppsummering

- Disse eksemplene, og de tilhørende konstruksjonene ved rekursjon og bevisene ved induksjon er det en fordel å kjenne til, og slike ting kan bli etterspurt til eksamen.
- Et av eksemplene vi ga på bruk av induksjonsbevis er at binomialkoeffisienten

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

forteller oss hvor mange delmengder med k elementer det finnes av en mengde med n elementer.

- Dette er det første ikke-trivielle resultatet i det vi kaller kombinatorikk, og *kombinatorikk* er temaet for resten av denne forelesningen.

OVER TIL KAPITTEL 9

Kombinatorikk

- KOMBINATORIKK er vitenskapen om hvordan man svarer på spørsmål som “hvor mange er det?” uten å telle.

Eksempel.

- Spørsmål:
Når de syv rette lottotalene er trukket ut, hvor mange forskjellige rekker har seks rette?
- Svar:
Det er syv forskjellige måter å plukke ut seks rette fra de syv rette.
Det er 27 måter å velge ut det ene tallet som er feil.
Det gir $7 \cdot 27 = 189$ forskjellige rekker med seks rette.

Kombinatorikk

- Selv om sikkert også informatikere spiller Lotto, gir ikke dette eksemplet noen god forklaring på hvorfor informatikkstudenter bør lære seg kombinatorikk.
- Kombinatorikk inngår som et vesentlig element i sannsynlighetsteori.
- Kombinatorikk inngår også når man skal vurdere hvor lang tid et program trenger for å nå i mål og når man skal vurdere hvor stor lagringsplass man må sette av for at et program eller en programpakke skal få det nødvendige arbeidsrommet.
- Vi skal stort sett holde oss til den type resultater som står i læreboka, men vil forsøksvis gi eksemplene en informatikkvinkling, der det er naturlig.
- Selv for noen av disse eksemplene, er den direkte sammenhengen med informatikk litt påtatt.

Kombinatorikk

Eksempel.

- Lagring av data i forskjellige registre kan illustreres ved lagring av kuler i bokser.
- Et naturlig spørsmål vil da være hvor mange forskjellige måter dette kan gjøres på.
- I dette eksemplet skal vi anta at vi har tretten kuler og fire bokser.
- Hvis vi spør om på hvor mange måter vi kan fordele 13 kuler på fire forskjellige bokser, er det to mulige presiseringer:

Kombinatorikk

Eksempel (Tilfelle 1).

- Alle kulene er forskjellige.
- Da har vi 13 kuler, og vi har fire muligheter for plassering av hver kule.
- Det gir

$$4^{13}$$

mulige fordelinger.

Kombinatorikk

Eksempel (Tilfelle 2).

- Kulene er like, mens boksene fortsatt er forskjellige, kall dem A, B, C og D.
- La

xxxxxxxxxxxxxxxxxxxx

være en mengde med 16 elementer (en mindre enn antall kuler pluss antall bokser).

- Det finnes

$$\binom{16}{3}$$

måter å omgjøre tre x til X på.

Kombinatorikk

Eksempel (Tilfelle 2, fortsatt).

- Eksempel

xxxXxxxxXxxXxxxx

- I dette tilfellet plasserer vi tre kuler i A (foran første X), fire kuler i B (mellom første og andre X), to kuler i C (mellom andre og tredje X) og fire kuler i D (bak siste X).
- Alle plasseringer av de tre X 'ene gir oss en fordeling av kulene på de fire boksene.

Kombinatorikk

Eksempel (Tilfelle 2, fortsatt).

- Omvendt vil en fordeling av 13 kuler på boksene A, B, C og D gi oss en plassering av X 'ene.
- Har vi to kuler i A, to i B, fem i C og fire i D svarer det til

xxXxxXxxxxXxxxx.

Kombinatorikk

Merk.

- Vi kunne ha formulert problemet om antall fordelinger også i det tilfellet hvor det ikke er forskjell på boksene.
- Det krever imidlertid at vi går ut over læreboka i en retning som ikke er prioritert, så det skal vi la ligge.
- Vi kunne også ha sett på tilfellet der vi har seks hvite og syv røde kuler.
- Dette er en utfordring dere bør kunne håndtere selv etter at forelesningen er over.

Kombinatorikk

- Det vi har lært av Tilfelle 2 i eksemplet over er at hvis vi skal fordele n identiske enheter på k forskjellige beholdere, finnes det

$$\binom{n+k-1}{k-1} = \frac{(n+k-1)!}{n! \cdot (k-1)!}$$

måter å gjøre det på.

- Vi skal se på et eksempel til.

Kombinatorikk

Eksempel.

- To programmerere skal utarbeide et system for lagring av enkle data og senere statistisk analyse av disse dataene.
- Man regner med å skulle registrere 1000 enkeltdata, og den ene programmereren lar en random-generator fordele disse dataene på fem ulike registre.
- Den andre programmereren er avhengig av å vite hvor mange dataenheter som er lagret i hvert enkelt register for å kjøre statistikkpakkene sine, og påpeker at det er ufattelig mange måter de innkomne dataene kan fordeles på.

Kombinatorikk

Eksempel (Fortsatt).

- Vedkommende har rett, for det eksakte tallet er

$$\binom{1000+4}{4} = \frac{1004!}{1000! \cdot 4!} = \frac{1004 \cdot 1003 \cdot 1002 \cdot 1001}{4 \cdot 3 \cdot 2}$$

måter.

- Det er i overkant av førti milliarder måter å fordele disse tusen dataenhetene på

Kombinatorikk

- Vi skal komme tilbake til opptelling av mulige fordelinger i forskjellige situasjoner.
- Først skal vi imidlertid se på en sammenheng mange kjenner fra sannsynlighetsteorien.
- I Læreboka går den under betegnelsen Inklusjons- og eksklusjonsprinsippet.

Kombinatorikk

Eksempel.

- På en skole er det 177 elever som driver aktiv idrett.

103 av elevene er aktive om sommeren og 85 av elevene er aktive om vinteren.

Det betyr at det er 188 aktiviteter fordelt på 177 elever.

For at dette skal stemme, må 11 av elevene drive både sommer- og vinteridrett, siden det er 11 flere aktiviteter enn det er elever.

- Hvis vi lar S være mengden av elever som driver sommeridrett og V være mengden av elever som driver vinteridrett, ser vi at
 - $|S| = 103$
 - $|V| = 85$
 - $|S \cup V| = 177$
 - $|S \cap V| = 11$

Kombinatorikk

Eksempel (Fortsatt).

- Det siste tallet regnet vi ut på grunnlag av de tre første.
- Vi ser at $|S| + |V| = |S \cup V| + |S \cap V|$ fordi det at det er flere aktiviteter enn elever skyldes at noen driver to aktiviteter, og differensen mellom antall aktiviteter og antall elever må være nøyaktig antallet på de som driver både sommer- og vinteridrett.

Kombinatorikk

Teorem (Inklusjons- og eksklusjonsprinsippet).

La A og B være to endelige mengder.

Da er $|A \cup B| = |A| + |B| - |A \cap B|$.

Bevis.

Hvis vi først teller opp elementene i A og deretter elementene i B , har vi talt elementene i $A \cap B$ to ganger.

For å få antall elementer i $A \cup B$ må vi derfor trekke fra det vi har talt for mye, nemlig antallet i $A \cap B$.

Kombinatorikk

Merk.

- Det er en nær sammenheng mellom *inklusions-* og *eksklusionsprinsippet* og en tilsvarende lov om sannsynlighet:

$$P(A) + P(B) = P(A \cup B) + P(A \cap B)$$

når A og B er uavhengige hendelser og P måler en sannsynlighet.

- Begge lovene illustreres greit med et Venn-diagram, hvor man ser at hvis vi skraverer sirkelskiven som markerer A i en retning og sirkelskiven som markerer B i en annen retning, er det akkurat feltet som markerer $A \cap B$ vi skraverer to ganger.

Kombinatorikk

Eksempel.

- Anta at vi har fått følgende oppgave:
Av 231 studenter var det 174 som greide oppgave 1 og 175 som greide oppgave 2.
Alle studentene greide minst en oppgave.
Hvor mange studenter greide begge oppgavene?

Kombinatorikk

Eksempel (Fortsatt).

- Løsning:
La A være mengden av studenter som greide oppgave 1 og B mengden av studenter som greide oppgave 2.
Da er $|A \cup B| = 231$, $|A| = 174$ og $|B| = 175$.
Inklusjons- og eksklusjonsprinsippet sier oss at

$$231 = 174 + 175 - |A \cap B|.$$

Det gir oss at

$$|A \cap B| = 174 + 175 - 231 = 118.$$

Det var 118 studenter som greide begge oppgavene.

Kombinatorikk

Eksempel.

La oss se på følgende oppgave:

- Medlemmene i et idrettslag blir bedt om å registrere seg på nettet med navn, adresse og enten e-postadresse eller mobilnummer.

Ledelsen ønsker å automatisere utsendelsen av informasjon, uten å bruke to informasjonskanaler til samme medlem, men av de 728 medlemmene er det 94 som har oppgitt både e-postadresse og mobilnummer.

Når vi vet at 562 medlemmer oppga e-postadresse, hvor mange oppga da mobilnummer?

Kombinatorikk

Eksempel (Fortsatt).

- Løsning:

La A være mengden av medlemmer som oppga e-postadresse og B være mengden av de som oppga mobilnummer.

Da sier inklusjons- og eksklusjonsprinsippet at

$$728 = 562 + |B| - 94$$

så

$$|B| = 728 + 94 - 562 = 260.$$

Det var 260 medlemmer som oppga mobilnummer.

Kombinatorikk

- Det neste prinsippet for beregning av antall muligheter vi skal se på er multiplikasjonsprinsippet.
- Multiplikasjonsprinsippet sier at hvis vi skal treffe en serie uavhengige valg, vil det totale antall muligheter være produktet av antall muligheter ved hvert valg.
- Igjen finnes det en klar parallell i sannsynlighetsteori, hvor vi finnes sannsynligheten for at en serie uavhengige hendelser finner sted ved å ta produktet av sannsynlighetene for enkelthendelsene.
- Vi skal illustrere dette prinsippet ved et par eksempler.

Kombinatorikk

Eksempel.

- Et norsk registreringsnummer for bil består av to store bokstaver og fem sifre.

- Vi bruker ikke bokstavene G, I, O, Q, Æ, Ø eller Å, fordi de enten ikke forekommer utenlands, eller fordi de kan forveksles med tall eller andre bokstaver.
- Da står vi igjen med $22 \cdot 22 = 484$ bokstavkombinasjoner.
- Første siffer i nummeret må være et av de ni tallene 1, 2, 3, 4, 5, 6, 7, 8, 9, mens de fire andre sifrene kan hentes fra alle de ti tallsymbolene.
- Det gir tilsammen

$$22 \cdot 22 \cdot 9 \cdot 10 \cdot 10 \cdot 10 \cdot 10 = 43.760.000$$

mulige registreringsnummere på norske biler.

Kombinatorikk

Eksempel (Fortsatt).

- Svenskene bruker tre bokstaver og tre tall, og hadde de begrenset seg til de bokstavene vi bruker i Norge, ville de kunne registrert færre biler.

Oppgave.

Hvor mange bokstaver må svenskene tillate for å kunne registrere like mange biler (eller fler) enn det nordmennene kan?

Kombinatorikk

Eksempel.

- Amerikanske lærebokforfattere lever i den tro at amerikanske collegestudenter lever en ikke ubetydelig del av livet sitt med å spise sammen.
- Derfor er følgende oppgave typisk for amerikanske lærebøker i diskret matematikk.
- En sandwich-bar tilbyr:
 1. Fire typer brød: Fint, mellomgrovt, grovt og glutenfritt.
 2. Tre typer smøring: Smør, majones og sennep.
 3. Seks typer hovedpålegg: Kalkun, roastbeef, skinke, tunfisk, skalldyr og soyaprotein.
 4. Fire typer tilbehør: Stekt bacon, salat, agurk og tomat.
 5. Tre valg på dressing, Thousen Islands, tomatdressing eller hvitløksdressing.
- Hvor mange forskjellige sandwicher er det mulig å komponere?

Kombinatorikk

Eksempel (Fortsatt).

Selv om ikke alle sammensetningene vil være like vellykkede rent smaksmessig, er det ingen føringer på hvilke valg som kan kombineres.

Da finner vi det totale antall muligheter ved å bruke multiplikasjonsprinsippet.

- Vi har da

$$4 \cdot 3 \cdot 6 \cdot 4 \cdot 3 = 864$$

forskjellige sammensetninger.

Kombinatorikk

Vi har nå gjennomgått nok teori til at følgende oppgave skal være lett:

Oppgave.

- a) Vi skal fordele syv like hvite kuler og seks like røde kuler på fire forskjellige bokser. Hvor mange måter kan dette gjøres på?
- b) Hvis vi krever at de hvite kulene skal ligge i de tre første boksene og de røde i de tre siste, hvor mange mulige fordelinger har vi da?
- c) Løs a) hvis vi i utgangspunktet bare hadde tre bokser, og sammenlikn svaret med svaret fra b).
Forklar det du observerer.

Kombinatorikk

- Det neste vi skal se på er hva vi mener med en permutasjon og på hvordan vi kan telle opp antall permutasjoner av en ordnet mengde.
- En permutasjon er en endring av en rekkefølge, eller en omstokking.
- Når vi stokker en kortstokk er poenget at kortene skal ligge i en annen rekkefølge, og med en fremmedord kan vi si at vi permuterer kortene.
- Vi skal se på noen eksempler.

Kombinatorikk

Eksempel.

- På hvor mange forskjellige måter kan vi skrive tallene 1, 2 og 3 i rekkefølge?
- Vi har tre valg for hvilket tall vi vil skrive først: 1, 2 eller 3.

- For hvert av disse valgene har vi to valg for hvilket som blir det neste tallet; Starter vi med 1 må det neste tallet være 2 eller 3, starter vi med 2 må det neste tallet være 1 eller 3 og starter vi med 3 må det neste tallet være 1 eller 2.
- Har vi bestemt hvilke to tall vi skriver først, gir det siste tallet seg av seg selv.
- Det finnes altså $3 \cdot 2 = 6$ måter å skrive disse tre tallene i rekkefølge på.

Kombinatorikk

Eksempel.

- Hvis vi utvider eksemplet vårt fra forrige side til å omfatte tallene 1, 2, 3 og 4 vil antall permutasjoner vokse til $4! = 24$ og tar vi med 5 i tillegg er antallet $5! = 120$.
- I det siste tilfellet har vi først fem valg for hvilket tall som skal skrives først, deretter fire valg for tall nr. 2, tre valg for tall nr. 3 og to valg for tall nr. 4. Det siste tallet gir seg selv.
- Generelt finnes det $n!$ permutasjoner av tallene $1, \dots, n$.
- Dette svarer også til hvor mange rekkefølger vi kan sette n elementer i. Eksempelvis kan syv studenter ordnes på $7! = 6720$ måter.

Kombinatorikk

Eksempel.

- Et kjent problem i litteraturen er Den handelsreisendes problem (The traveling salesman).
- Hvis vi har gitt n byer som skal besøkes, og vi kjenner avstanden mellom to og to av byene, hva er da den korteste veien gjennom alle byene?
- Det er ennå ingen som har kommet opp med et program som løser dette problemet når antall byer er stort, som for eksempel alle tettsteder i Norge med mer enn 300 innbyggere.
- Vi skal se på hva dette problemet kan ha med antall permutasjoner å gjøre.

Kombinatorikk

Eksempel (Fortsatt).

Hvis antall byer som skal besøkes er mindre, blir selvfølgelig oppgaven gjennomførbar. Anta at vi har fått i oppdrag å skrive et program som finner den korteste reiseruten fra by A til by B, og som går gjennom ti andre byer C_1, \dots, C_{10} i en eller annen rekkefølge.

Igjen kan vi anta at alle avstander er kjent.

En måte å gjøre dette på er å liste opp alle mulige rekkefølger vi kan besøke byene C_1, \dots, C_n i, regne ut alle reiselengdene og så velge ut den korteste.

Problemet er at det finnes $10! = 3.628.800$ forskjellige rekkefølger vi kan velge mellom.

Kombinatorikk

Eksempel (Fortsatt).

- Det betyr altså at det finnes over tre og en halv million måter å reise fra Oslo til Kirkenes på, når reisen skal gå via
 - Kristiansand
 - Stavanger
 - Bergen
 - Molde
 - Kristiansund
 - Trondheim
 - Bodø
 - Narvik
 - Tromsø
 - Alta

Kombinatorikk

Eksempel (Fortsatt).

Øker vi antall byer som skal besøkes til 12, hvis vi for eksempel vil besøke Haugesund og Levanger i tillegg, vil vi være i nærheten av 400.000.000 enkeltruter, og da begynner de raske maskinene å slite.

Det vil gå flere generasjoner maskiner mellom hver gang vi kan øke antall byer med 1 hvis vi bruker denne naive måten.

Kombinatorikk

Eksempel (Fortsatt).

- Det man i praksis gjør er å akseptere at det er dumt å bruke år på å finne ut av om man kan spare noen få kilometers reise, og utvikler raske algoritmer som gir effektive reiseruter, uten å garantere at den finner den mest effektive.

- Det finnes elektroniske reiseplanleggere som må forene hensynet til kort regnetid og et godt resultat.
- Tilsvarende optimeringsproblemer finner man for effektiv utnyttelse av lagerplass, effektiv organisering av produksjonsleddene i en bedrift og liknende.

Kombinatorikk

Eksempel.

- Dette eksemplet er stjålet fra en tidligere lærebok i diskret matematikk.
- Hvor mange ord kan vi skrive ved hjelp av bokstavene i
MISSISSIPPI?
- Det er 11 bokstaver, og har vi en blytype for hver bokstav, kan vi sette disse i 11! forskjellige rekkefølger.

Kombinatorikk

Eksempel (Fortsatt).

- Rekkefølgen vi setter de to P'ene i, betyr imidlertid ikke noe for resultatet. Det alene halverer antall ord vi kan skrive.
- Det er fire I'er og fire S'er. Den innbyrdes rekkefølgen blant I'ene og blant S'ene betyr heller ikke noe for hvordan det ferdige ordet ser ut.
- Det er $4! = 24$ måter å trykke de fire S'ene og $4! = 24$ måter å trykke de fire I'ene på.
- Det betyr at antall forskjellige ord vi kan skrive er

$$\frac{11!}{4! \cdot 4! \cdot 2!} = 34650.$$

Kombinatorikk

Oppgave.

Hvor mange forskjellige ord kan vi skrive ved å stokke om på bokstavene i ordet

PUSLESPILL

Regn ut svaret fullstendig.

Forelesning 20

Kombinatorikk

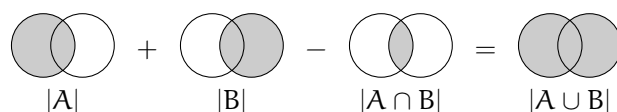
Roger Antonsen - 7. april 2008

Kombinatorikk

- Kombinatorikk er studiet av *opptellinger, kombinasjoner og permutasjoner*.
- Vi finner svar på spørsmål "Hvor mange måter ...?" uten å telle.
- Viktig del av f.eks. kompleksitetsanalyse av algoritmer.
 - Hvor mye *tid* bruker en algoritme?
 - Hvor mye *plass* bruker en algoritme?
- Grunnleggende, nyttig og fascinerende matematikk som dere må beherske.
- Vi skal i dag gjøre oss ferdige med kapittel 9.
- Først litt repetisjon.

Repetisjon

- Inklusjons- og eksklusjonsprinsippet:



- Hvis vi først teller opp elementene i A og deretter elementene i B, har vi talt elementene i $A \cap B$ to ganger.
- For å få antall elementer i $A \cup B$ må vi derfor trekke fra det vi har talt for mye, nemlig antallet i $A \cap B$.

Repetisjon

- Multiplikasjonsprinsippet:
Hvis vi skal treffe en serie uavhengige valg, vil det totale antall muligheter være produktet av antall muligheter ved hvert valg.

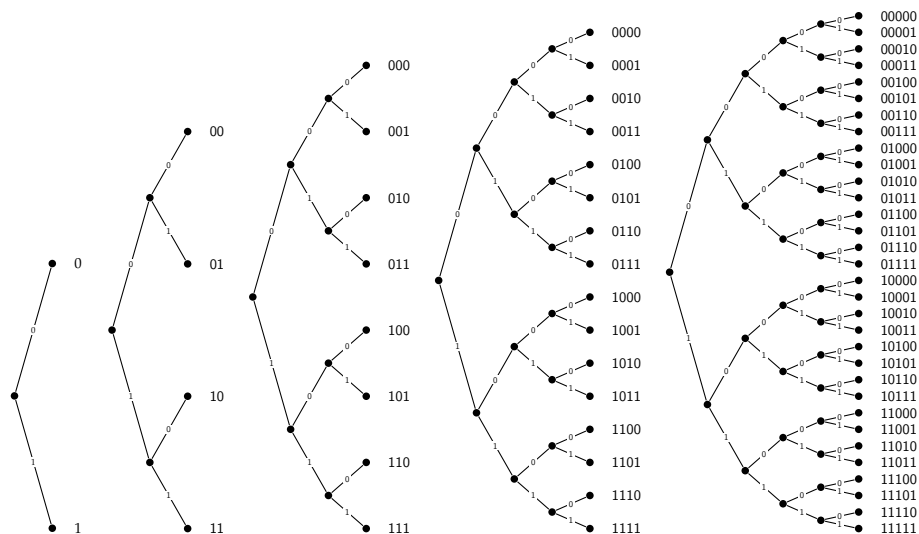
$$|A \times B| = |A| \cdot |B|$$

Antall elementer i det kartesiske produktet $A \times B$ er antall elementer i A multiplisert med antall elementer i B.

- Begge prinsippene kan generaliseres til flere enn to mengder.
 - Generaliseringen av inklusjons- og eksklusjonsprinsippet ses lett ved hjelp av Venn-diagrammer.
 - Generaliseringen av multiplikasjonsprinsippet blir

$$|A_1 \times A_2 \times \cdots \times A_n| = |A_1| \cdot |A_2| \cdot \cdots \cdot |A_n|$$

Eksempel - det er 2^n binære tall av lengde n

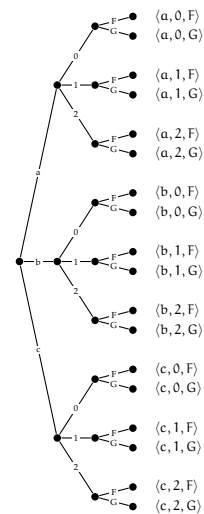


Eksempel - $\{a, b, c\} \times \{0, 1, 2\} \times \{F, G\}$

Multiplikasjonsprinsippet gir oss følgende:

$$\begin{aligned}
 & |\{a, b, c\} \times \{0, 1, 2\} \times \{F, G\}| \\
 &= |\{a, b, c\}| \cdot |\{0, 1, 2\}| \cdot |\{F, G\}| \\
 &= 3 \cdot 3 \cdot 2 \\
 &= 18
 \end{aligned}$$

Vi kan illustrere det slik:



Mer repetisjon

Definisjon (Permutasjon).

En *permutasjon* er en endring av rekkefølgen av elementene i en ordnet mengde. Vi sier også at en *permutasjon* av en mengde er en ordning av elementene i mengden.

Eksempel.

Permutasjonene av $\{A, B, C\}$ er ABC, ACB, BAC, BCA, CAB, CBA.

- Det er $n!$ permutasjoner av en mengde med n elementer.
- Og vi vet (selvfølgelig) at $n! = n \cdot (n - 1) \cdot (n - 2) \cdots 1$
- I eksempelet har vi 3 elementer og $3! = 3 \cdot 2 \cdot 1 = 6$ permutasjoner.

Plan for dagen

- Mer om permutasjoner og ordnet utvalg ${}^n P_r$
- Mer om kombinasjoner $\binom{n}{r}$ "n velg r"
- Binomialkoeffisientene
- Pascals trekant
- Generalisering av første kvadratsetning $(a + b)^2 = a^2 + 2ab + b^2$
- Oppsummering av kombinatoriske prinsipper
- Eksempler og oppgaver

Ordnet utvalg

Vi skal nå se på det som kalles ordnet utvalg fra en mengde.

Eksempel.

Oppgave:

I et barneskirenn er det med 20 barn.

Det er lov til å opplyse om hvem som tok de tre første plassene, mens resten ikke skal rangeres.

Hvor mange forskjellige resultatlister kan man få?

Eksempel (Fortsatt).

Løsning:

Det fins 20 mulige vinnere, deretter 19 mulige andre plasser og til sist 18 mulige tredjeplasser.

Det fins altså $20 \cdot 19 \cdot 18 = 6840$ forskjellige resultatlister.

- Legg merke til at

$$20 \cdot 19 \cdot 18 = \frac{20 \cdot 19 \cdot 18 \cdot 17 \cdot 16 \cdots 2 \cdot 1}{17 \cdot 16 \cdots 2 \cdot 1} = \frac{20!}{(20-3)!}$$

- Vi skal nå definere dette mer generelt og bruke notasjonen ${}^{20}P_3$ for dette tallet.

Definisjon.

- La r og n være naturlige tall slik at $r \leq n$.
- Med ${}^n P_r$ mener vi $\frac{n!}{(n-r)!}$

Merk.

- ${}^n P_r$ forteller oss hvor mange måter vi kan trekke r elementer i *rekkefølge* ut fra en mengde med n elementer på.

- Når $n = r$ bruker vi at $0! = 1$. Da får vi

$${}^n P_n = \frac{n!}{(n-n)!} = \frac{n!}{0!} = \frac{n!}{1} = n!$$

- Det er som forventet, siden det er $n!$ permutasjoner av en mengde med n elementer i.

Eksempel.

- En idrettsleder har syv løpere i stallen sin, og skal velge ut fire av dem til å delta i en stafett.
- I et stafettlag spiller rekkefølgen stor rolle, især om idrettsgrenen er langrenn og det er to etapper i klassisk og to i fristil.
- Da er det ${}^7 P_4 = \frac{7!}{3!} = 7 \cdot 6 \cdot 5 \cdot 4 = 840$ forskjellige mulige laguttak.
- Det er et under at avisene mener seg å vite hva uttaket vil bli dagen i forveien.


Kombinasjoner

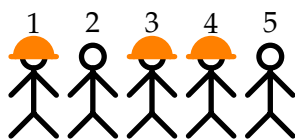
$$\binom{n}{k}$$

angir hvor mange delmengder med k elementer det finnes av en mengde med n elementer

- Vi har tidligere vist dette ved induksjon.
- Det er også mulig å vise dette rent kombinatorisk, som vi skal gjøre snart.
- Slike tall kalles blant annet for *binomialkoeffisienter*.

Eksempel

- Anta at vi skal fordele tre oransje hatter, , på fem barn.



- Hver kombinasjon svarer til en delmengde av $\{1, 2, 3, 4, 5\}$.
- Antall måter å velge tre barn på *i rekkefølge* er ${}^5P_3 = 5 \cdot 4 \cdot 3 = 60$.
- Her vil hver kombinasjon av hatter, f.eks. $\{1, 3, 4\}$, bli telt $3! = 6$ ganger, som 134, 143, 314, 341, 413, og 431.
- Hvis vi skal ta høyde for dette, så må vi dele på 6.
- Antall måter å fordele hattene er derfor $60/6 = 10$, som er $\binom{5}{3}$.

Teorem.

- La A være en mengde med n elementer, og la $0 \leq k \leq n$.
- Da finnes det

$$\binom{n}{k}$$

forskjellige delmengder B av A .

Bevis (Nytt, og fritt for induksjon).

- Antall måter å velge k elementer *i rekkefølge* fra A på er

$${}^n P_k = \frac{n!}{(n-k)!}$$

- For hver delmengde B med k elementer, så fins det $k!$ forskjellige ordnede utvalg fra A som gir oss B .
- Da må antall mengder B med k elementer være

$$\frac{{}^n P_k}{k!} = \frac{n!}{(n-k)! \cdot k!} = \binom{n}{k}$$

- Legg merke til at

$$\binom{n}{k} = \binom{n}{n-k}$$



- Hvorfor det?

- For eksempel, hvis vi har en mengde med 20 elementer, så er det $\binom{20}{18}$ delmengder med 18 elementer.
- For hver slik mengde, så har vi også en mengde med 2 elementer.
 - Vi kan f.eks. lage en funksjon som til enhver delmengde av størrelse 18 gir en delmengde av størrelse 2.
 - Denne funksjonen vil være både surjektiv og injektiv.
- Derfor har vi at $\binom{20}{18} = \binom{20}{2} = \frac{20 \cdot 19}{2 \cdot 1} = 190$.
- Antall delmengder av størrelse k må være lik antall delmengder av størrelse $n - k$.
- Det er like mange måter å velge n elementer på som det er å måter å *velge bort* n elementer på.

Binomialkoeffisientene

- Tallene $\binom{n}{k}$ kalles blant annet for *binomialkoeffisienter*.
- Følgende (rekursive) sammenheng var utgangspunktet for et tidligere induksjonsbevis:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

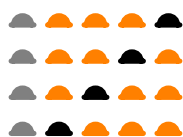
- Hvorfor er det slik? La oss se på et eksempel.
- Tre, , av fem hatter, , skal være oransje.
- Hvor mange måter kan dette gjøres på?

$$\binom{5}{3} = \binom{4}{2} + \binom{4}{3}$$

- Hvis den første hatten er oransje, så må to av de fire resterende hattene være oransje. Det er $\binom{4}{2} = 6$ måter å gjøre dette på.



- Hvis den første hatten er svart, så må *tre* av de fire resterende hattene være oransje. Det er $\binom{4}{3} = 4$ måter å gjøre dette på.



- Hvis vi multipliserer dette ut, så får vi 2^n ledd.
- Hvert ledd består av n faktorer, hvor hver faktor er enten a eller b , f.eks. $abaa \cdots b$.
- Hvis $B \subseteq A = \{1, \dots, n\}$, så lar vi B svare til leddet hvor faktor nummer i er b hvis $i \in B$ og a ellers.
- F.eks. vil $\{1, 4, 5\}$ svare til leddet $\underset{1}{b}\underset{2}{a}\underset{3}{a}\underset{4}{b}\underset{5}{b}\underset{6}{a} \cdots \underset{n}{a}$
- Hvert ledd kommer fra en og bare en mengde B ; det vi har beskrevet er en surjektiv og injektiv funksjon fra potensmengden av A til leddene vi får når vi regner ut $(a + b)^n$.

Bevis (Fortsatt).

- Det fins $\binom{n}{k}$ delmengder av A med k elementer.
- Da fins det $\binom{n}{k}$ ledd med k b 'er og $n - k$ a 'er.
- Disse leddene ordnes til

$$\binom{n}{k} a^{n-k} b^k$$

- Dette er nøyaktig leddet med indeks k i teoremet.
- Siden k er vilkårlig må formelen i teoremet gi oss verdien på $(a + b)^n$.
- Dette avslutter beviset.

Oppsummering av regneprinsipper

- Ordnet utvalg med repetisjon: n^r
 - Hvor mange binære tall av lengde 5 fins det?
 - Det er $2^5 = 32$.
- Ordnet utvalg uten repetisjon: ${}^n P_r$
 - På hvor mange måter kan vi trekke to kort fra en kortstokk?
 - Det er ${}^{52} P_2 = 52 \cdot 51 = 2652$.
- Permutasjoner: $n!$
 - På hvor mange måter kan vi stokke om ordet LAKS?
 - Det er $4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$.
- Kombinasjoner: $\binom{n}{k}$
 - Hvor mange delmengder av $\{a, b, c, d, e\}$ har to elementer?
 - Det er $\binom{5}{2} = \frac{5 \cdot 4}{2 \cdot 1} = 10$
- Vi skal se på noen flere eksempler.

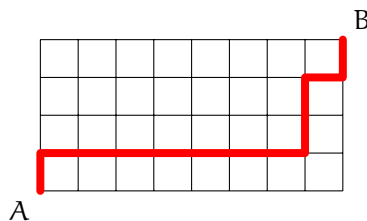
Først en liten digresjon om store tall

I kombinatorikk kommer vi fort opp i *veldig* store tall.

- Bredden til et hårstrå: 10^6 atomer.
- Atomer i en vanndråpe: 10^{21} atomer.
- Atomer i universet: 10^{80} atomer.
- Antall forfedre 265 generasjoner tilbake = antall atomer i universet.
- Og disse tallene er ganske små...
- Allikevel kan vi representere dem og regne på dem uten store problemer.

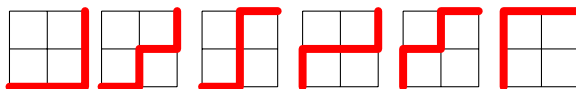
Eksempel

På hvor mange forskjellige måter kan vi gå fra A til B i dette 8 ganger 4-rutenettet? Vi må gå ett steg av gangen og kun oppover eller til høyre.



- Hvor mange slike stier er det?
- Denne stien kan representeres som $\uparrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \uparrow \uparrow \rightarrow \uparrow$.
- Dette er et ord på 12 tegn over alfabetet $\{\uparrow, \rightarrow\}$ hvor fire av tegnene er \uparrow og åtte av tegnene er \rightarrow .
- Hvor mange slike ord er det? Det er $\binom{12}{4} = \frac{12 \cdot 11 \cdot 10 \cdot 9}{4 \cdot 3 \cdot 2 \cdot 1} = 495$

- Vi sjekker at det stemmer for 2 ganger 2-rutenettet:



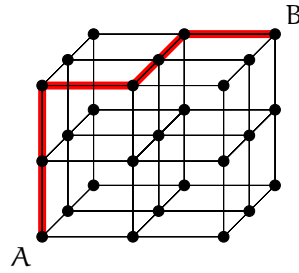
- Antall ord blir i dette tilfellet:

$$\binom{4}{2} = \frac{4 \cdot 3}{2 \cdot 1} = 6$$

som stemmer...

Oppgave

På hvor mange forskjellige måter kan vi gå fra A til B i denne $2 \times 2 \times 2$ -kuben? Vi må gå ett steg av gangen og kun oppover, til høyre eller innover?



Eksempel

- (Dette ble også nevnt på forrige forelesning i et introduksjonseksempel om kuler og bokser.)
- På hvor mange ulike måter kan de 6 karakterene A til F gis til 10 studenter?
- Vi ikke er interessert i hvilken karakter en bestemt student får, men kun antallet av hver karakter i fordelingen.
- Det er 6 muligheter for hver av de 10 studentene.
- Kan vi bruke multiplikasjonsprinsippet og si at svaret er 6^{10} ? **Nei**
- La oss lage 6 båser og putte studentene i hver sin bås avhengig av hvilken karakter de får.

Eksempel

- 2 studenter per karakter, og ingen stryk kan representeres slik:



- Fordelingen 123211 kan representeres slik:



- At alle stryker kan representeres slik:



- Hvor mange slike kombinasjoner fins det?
- Av 15 tegn må 5 være røde streker.
- Antall muligheter må være

$$\binom{15}{5} = \frac{15 \cdot 14 \cdot 13 \cdot 12 \cdot 11}{5 \cdot 4 \cdot 3 \cdot 2 \cdot 1} = 3003$$

Eksempel

- Mer generelt har vi

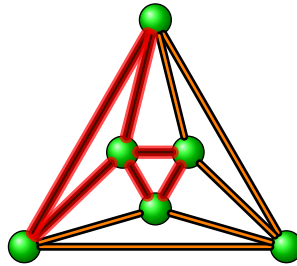
$$\binom{n+k-1}{k-1}$$

som gir hvor mange forskjellige måter vi kan fordele n identiske elementer i k forskjellige beholdere på.

- Dette kalles også for uordnet utvalg *med repetisjon*.

Grafteori

- Neste gang begynner vi med *grafteori*.
- En *graf* består av *noder* og *kanter*:



- Oppgave: klarer dere å tegne denne på et ark uten å løfte blyanten og uten å gå over en kant to ganger?

Forelesning 21

Grafteori

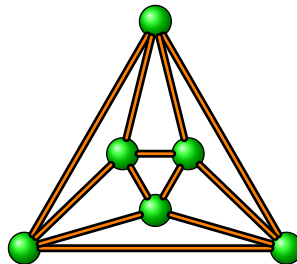
Roger Antonsen - 9. april 2008

Introduksjon

- Vi skal nå over til kapittel 10 & grafteori.
- Grafer fins overalt rundt oss!
- Grafteori er en viktig del av både *anvendt* og *teoretisk* matematikk.
- Grafteori brukes til å modellere problemer innenfor mange områder: informatikk, biologi, samfunnsvitenskap, lingvistikk, fysikk ...
- Mange problemer kan *representeres* ved å bruke grafer.
- Vi har møtt idéen om *representasjon* flere ganger allerede.
- Vi representerer f.eks. reelle tall ved hjelp av binære tall.
- Vi kan representere et matematisk problem som et annet som er enklere å løse.
- En representasjon gjør at vi kan se bort fra det som er irrelevant. Vi fanger inn *essensen*.
- Det er akkurat det som skjer i grafteori.

En graf

- En *graf* består av *noder* (●) og *kanter* (—).
- Dette er et eksempel på en graf:



- Vi avsluttet forrige gang med en oppgave: klarer dere å tegne denne på et ark uten å løfte blyanten og uten å gå over en kant to ganger?
- Etter disse forelesningene i grafteori skal alle klare å besvare dette spørsmålet umiddelbart.
- Vi skal se at oppgaven er ekvivalent med å finne en såkalt *Eulersti*.

Søkealgoritmer for grafer

- Søkealgoritmer for grafer er et viktig tema.

- Kjente søkealgoritmer for grafer er f.eks.:
 - *Prims algoritme* for å finne et *minimalt spennetre* i en vektet graf. (Kruskals algoritme for samme problem.)
 - *Dijkstras algoritme* for å finne *minste avstand*, eller *korteste sti*, i en vektet graf.
- Vi kan søke bredde først eller dybde først
- For veldig mange grafproblemer har man ikke funnet *effektive* algoritmer.

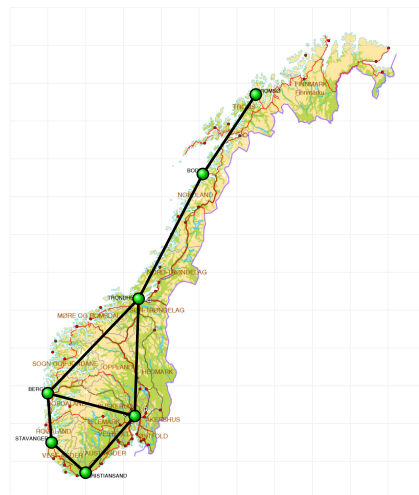
Eksempler på grafer

Grafer kan representere mange forskjellige ting.

- Nodene kan være studentene på Blindern, og kantene kan representere at to kjenner hverandre.
- Nodene kan være tilstandene som et dataprogram er i, og kantene kan representere overganger mellom tilstandene.
- En graf kan representere lenkestrukturen på et nettsted, hvor nodene er nettsider og kantene er lenker.
- Listen fortsetter: elektroniske kretser, molekyler i kjemi, datanettverk, analyse av nettverkstrafikk...
- Et *tre* er en spesiell type graf. Vi kommer til trær i kapittel 11.
- Vi skal se på flere eksempler på grafer før vi begynner med teorien.

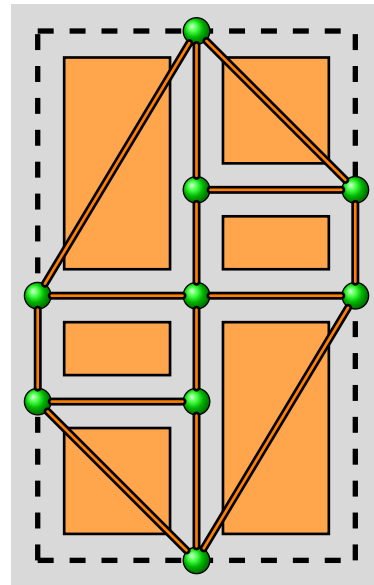
Kart som grafer

- Et kart kan gi utgangspunkt for flere forskjellige grafer.
- En mulighet er at
 - nodene representerer *byer*
 - kantene representerer *veier*
- En annen mulighet er at
 - nodene representerer *områder*, f.eks. fylker
 - kantene representerer *grenser*
- Når vi har representasjonen, så kan vi egentlig glemme det opprinnelige kartet.



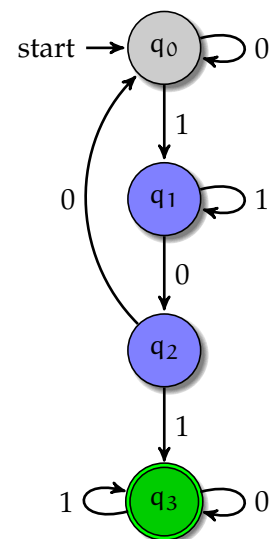
Veinett som grafer

- Et veinett kan representeres som en graf.
- Vi kan la hvert *kryss* svare til en node.
- Vi kan la *veiene* som forbinder kryssene svare til kantene.
- Når vi har tegnet opp grafen, så kan vi resonnerer om den i stedet for om selve veinettet.



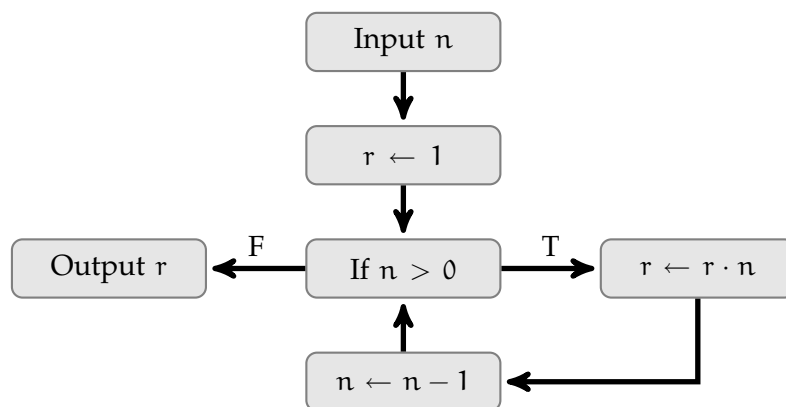
Endelige tilstandsmaskiner som grafer

- Vi kan tenke på *endelige tilstandsmaskiner* som grafer.
- Her kalles q_0, q_1, q_2 og q_3 *tilstander* og overgangene kalles *transisjoner*.
- Vi har en starttilstand, q_0 og en såkalt *aksepterende tilstand*, q_3 .
- Hvis vi begynner med tallet 1101 og følger kantene etter hvert som vi leser siffer, så ender vi opp i q_3 . Siden det er en aksepterende tilstand, er 1101 *akseptert*.
- Tallet 100 aksepteres ikke, siden vi ender opp i tilstand q_0 , som ikke er aksepterende.
- Ser du hvilke tall som aksepteres?



Flytdiagrammer som grafer

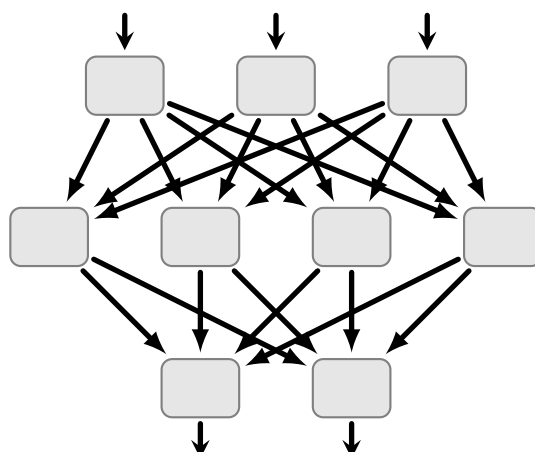
- Vi kan tenke på *flytdiagrammer* som grafer.
- I følgende eksempel representerer nodene programinstruksjoner.



- Hvilket program er dette?

Flytdiagrammer og multiplikasjonsprinsippet

- Multiplikasjonsprinsippet igjen.



Grafteori - definisjoner og begreper

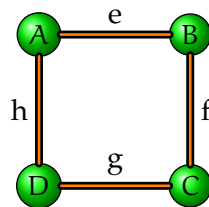
Definisjon (Graf).

En *graf* G består av en ikke-tom mengde *noder* V og en mengde *kanter* E , slik at enhver kant forbinder nøyaktig to noder med hverandre eller en node med seg selv.

- Dette er med vilje litt upresist.
- Vi presiser heller etter hvert når vi trenger det.
- På engelsk brukes begrepene
 - *vertex/vertices* om noder, og

– *edges* om kanter.

- Vi tegner noder slik: ●
- og kanter slik: —
- Det er ikke viktig akkurat *hvordan* vi tegner grafer; det er *strukturen* i graf som er viktig, hvilke noder som er forbundet med hvilke via en kant.



Her er A, B, C og D noder, mens e, f, g og h er kanter.

Definisjon (Inntil/naboer).

En kant ligger *inntil* (engelsk: *incident*) nodene som forbindes av den. To noder er *naboer* (engelsk: *adjacent*) hvis de forbindes av en kant.

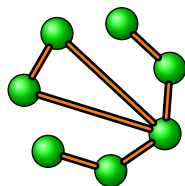
- Kanten e ligger inntil nodene A og B.
- Nodene B og C er naboer, siden de forbindes av kanten f.

Sammenhengende grafer

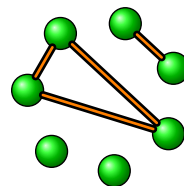
En graf trenger ikke å være *sammenhengende*.

Definisjon (Sammenhengende).

En graf er *sammenhengende* (engelsk: *connected*) hvis det er mulig å komme fra enhver node til enhver annen node ved å følge kantene.



En *sammenhengende* graf.



En *usammenhengende* graf.

Tomme grafer og løkker

En graf trenger ikke å ha noen kanter, men en må ha minst én node. Grafer uten kanter kalles *nullgrafer* eller *tomme grafer* (engelsk: *null graph*).



En tom graf.

En graf kan ha *løkker* (engelsk: *loop*), en kant som går fra en node til den samme noden.



En graf med en løkke.

Parallelle kanter og enkle grafer

En graf kan ha *parallelle* kanter, to eller flere kanter som forbinder de samme to nodene,



En graf med parallelle kanter.

Definisjon (Enkel).

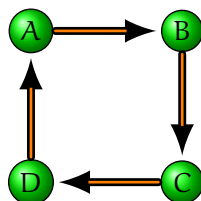
En graf er *enkel* (engelsk: *simple*) hvis den ikke har løkker eller parallelle kanter.

- Det er ganske vanlig å definere grafer slik at løkker og parallelle kanter ikke forekommer.

Rettete grafer

Definisjon.

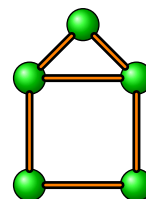
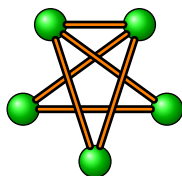
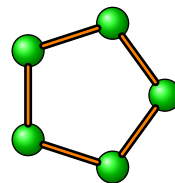
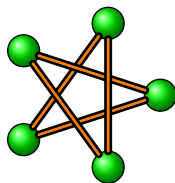
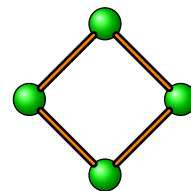
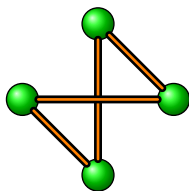
En *rettet graf* (engelsk: *directed*) er en graf hvor hver kant har en retning.

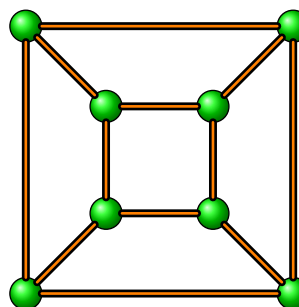
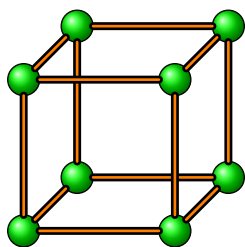


- En *relasjon* kan ses på som en rettet graf.
- Denne grafen svarer til relasjonen $\{\langle A, B \rangle, \langle B, C \rangle, \langle C, D \rangle, \langle D, A \rangle\}$
- Hvis vi har en *symmetrisk* relasjon, riktignok, så kan vi tenke på denne som en vanlige (urettet) graf.
- Foreløpig skal vi ikke snakke om rettede grafer.

Måter å tegne opp grafer på

- Det fins ikke noen unik måte å tegne opp en graf på.
- Vi bryr oss for eksempel ikke om hvor lange kantene er, om de er bøyd, etc.
- Det eneste som spiller noen rolle er om to noder er forbundet med en kant.
- La oss se på noen eksempler. Følgende par av grafer er identiske, men tegnet opp på forskjellige måter.
- Vi forestiller oss at kantene er elastiske og at vi flytter om på nodene.
- Vi skal etter hvert presisere dette gjennom begrepet *isomorfi*. Følgende par av grafer kalles *isomorfe*.



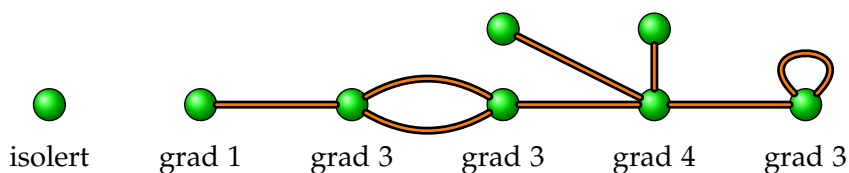


Grafteori

Graden til noder

Definisjon (Grad).

Graden (engelsk: *degree*) til en node v er antall kanter som ligger inntil v . En løkke teller som to kanter. Med $\text{deg}(v)$ mener vi graden til v . En node med grad 0 kalles *isolert*.

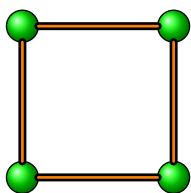


Teorem.

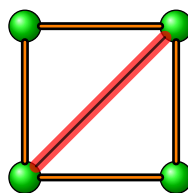
Summen av gradene til alle nodene i en graf er lik 2 ganger antallet kanter. Hvis V er mengden av noder og E er mengden av kanter, så har vi

$$\sum_{v \in V} \text{deg}(v) = 2|E|.$$

- Hver *kant* som legges til i en graf vil øke summen av gradene med to.
- La oss se på et eksempel.



Antall kanter er 4.
Summen av gradene er 8.



Antall kanter er 5.
Summen av gradene er 10.

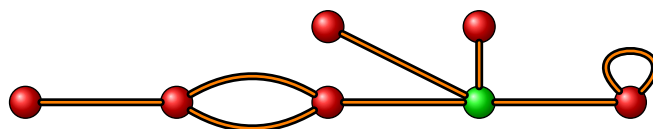
Bevis.

Hvis vi legger sammen gradene til alle nodene, så vil hver kant telle to ganger, siden hver kant ligger inntil to noder.

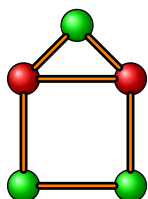
Håndhilselemmaet

Lemma (håndhilselemmaet).

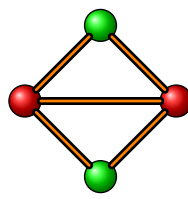
Det er alltid et partall antall noder av odde grad i en graf.



Her er det 6 noder (markert med rødt) med odde grad.



Her er det 2 noder av odde grad.



Her er det 2 noder av odde grad.

- Hvis vi forestiller oss mange mennesker samlet i et rom og at man håndhilser på hverandre, så må antallet av de som håndhilser på et odde antall personer være et partall.
- Vi kan representere denne situasjonen ved å representere menneskene som noder. En kant vil da representere at to personer håndhilser på hverandre.

- Det kalles et lemma fordi det ikke er så interessant i seg selv, men er nyttig for å bevise andre lemmaer og teoremer.
- Vi skal nå bevise håndhilselemmaet.

Bevis (håndhilselemmaet).

La G være en graf. Vi deler mengden V av noder inn to: V_o er de som har odde grad (de som var røde) og V_p er de som har lik grad (de som var grønne). Vi har vist et teorem som sier at summene av gradene til *alle* nodene er to ganger antall kanter.

$$\sum_{v \in V} \deg(v) = 2|E|$$

Siden vi har delt opp mengden av noder i to, kan vi skrive dette slik:

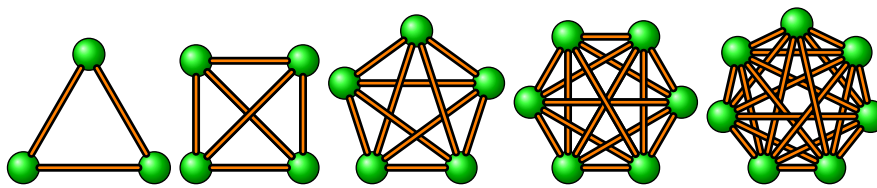
$$\sum_{v \in V_o} \deg(v) + \sum_{v \in V_p} \deg(v) = 2|E|$$

Siden $2|E|$ er et partall og summen av gradene til nodene i V_p er et partall, så må summen av gradene til nodene i V_o også være et partall. Siden hver node i V_o har odde grad, så må det være et partall antall av dem.

Komplette grafer

Definisjon (Komplett graf).

En enkel graf er *komplett* hvis hver node er nabo med enhver annen node.



Komplette grafer (K_3, K_4, K_5, K_6, K_7).

- Hvor mange kanter er det i en komplett graf?
- K_3 har 3 kanter. K_4 har 6 kanter. K_5 har 10 kanter. K_6 har 15 kanter. K_7 har 21 kanter. Er det noen som ser et mønster?

Teorem.

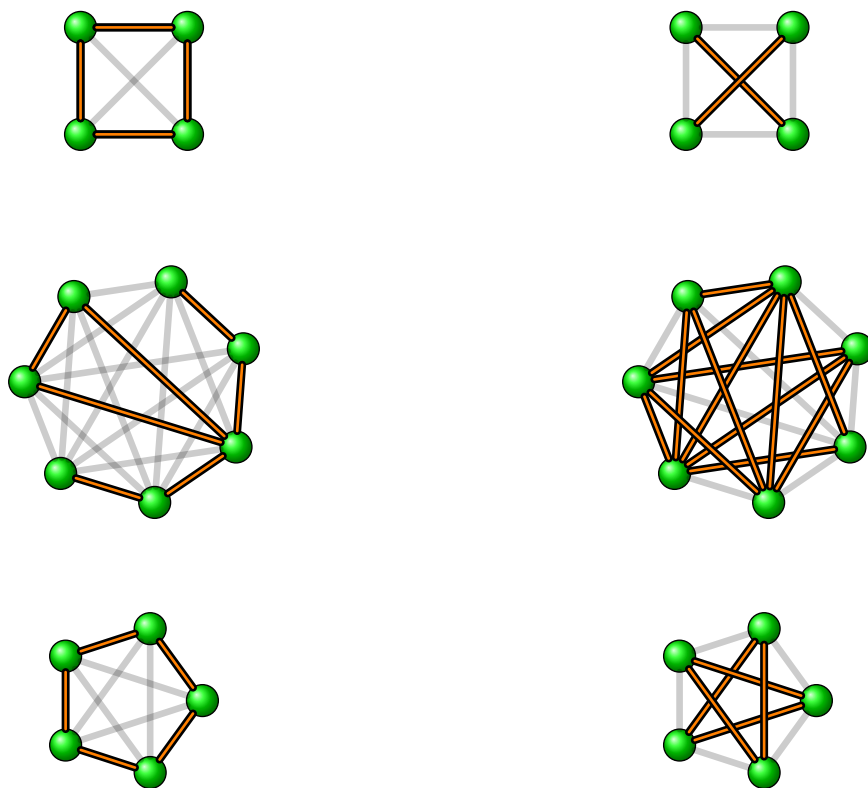
Det er $\binom{n}{2}$ kanter i en komplett graf med n noder.

Komplementet til en graf

Definisjon (Komplement).

La G være en enkel graf. Da er *komplementet* til G grafen som har de samme nodene som G , men hvor to noder er naboer hvis og bare hvis nodene *ikke* er naboer i G . Vi skriver \bar{G} for komplementet til G .

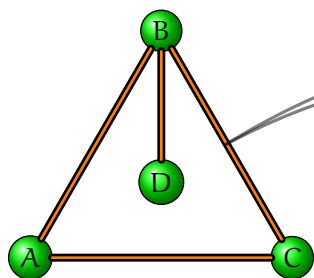
Vi skal se på noen grafer og deres komplementer.



- I det siste tilfellet fikk vi ikke noen *ny* graf når vi tok komplementet.
- Slike grafer kalles *selv-komplementære*.

Matriserepresentasjoner

På samme måte som med relasjoner, så har grafer en matriserepresentasjon. Vi kaller en slik matrise for en *koblingsmatrise* (engelsk: *adjacency matrix*).



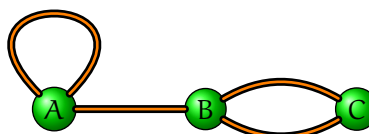
$$\begin{array}{c}
 A \quad B \quad C \quad D \\
 A \quad \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \\
 B \\
 C \\
 D
 \end{array}$$

Koblingsmatrisen til grafen.

Definisjon (Koblingsmatrise).

Hvis G er en graf med n noder, v_1, \dots, v_n , så er koblingsmatrisen til G en n -ganger- n -matrise hvor tallet i rad i og kolonne j er antall kanter mellom v_i og v_j .

$$\begin{array}{c}
 A \quad B \quad C \\
 A \quad \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 2 \\ 0 & 2 & 0 \end{bmatrix} \\
 B \\
 C
 \end{array}$$



Koblingsmatrisen til grafen.

- Legg merke til at vi kan speile en matrise om diagonalen.
- Det er fordi vi kun ser på *urettede* grafer.
- Hvis vi ser på *rettede* grafer, så kan vi ikke speile matrisen om diagonalen.
- Vi kunne også speil om diagonalen for *symmetriske relasjoner*.
- En forskjell mellom symmetriske relasjoner og grafer er at vi tillater *parallelle* kanter i grafene.
- De kan vi ikke fange inn ved hjelp av en relasjon.
- Det fins flere matriser for samme graf, avhengig av rekkefølgen vi gir nodene i.

Forelesning 22

Grafteori

Roger Antonsen - 14. april 2008

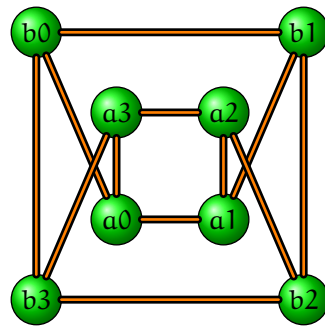
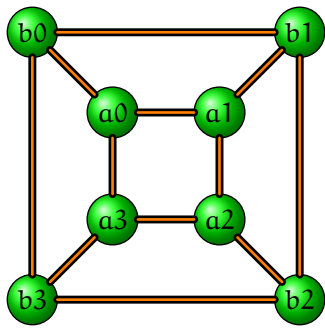
Repetisjon og mer motivasjon

- Først litt repetisjon
- En graf består av *noder* og *kanter*
- Kanter ligger *inntil* noder, og noder kan være *naboer*.
- Vi bør kjenne til begrepene om *sammenhengende grafer*, *tomme grafer*, *løkker*, *parallelle kanter*, *enkle grafer* og *komplette grafer*.
- Hver node har en *grad*.
 - Summen av gradene til alle nodene i en graf er lik 2 ganger antallet kanter.
 - Håndhilselemmaet: Det er alltid et partall antall noder av odde grad i en graf.
- Vi skal kjenne til *komplementet* av en graf og *matriserepresentasjoner*.

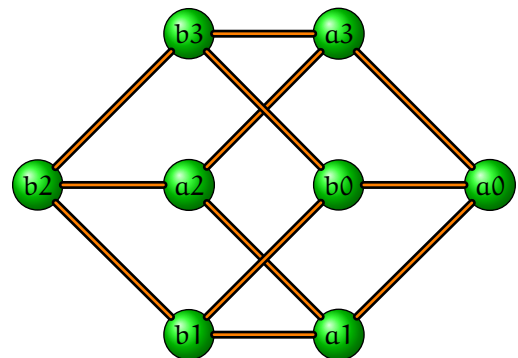
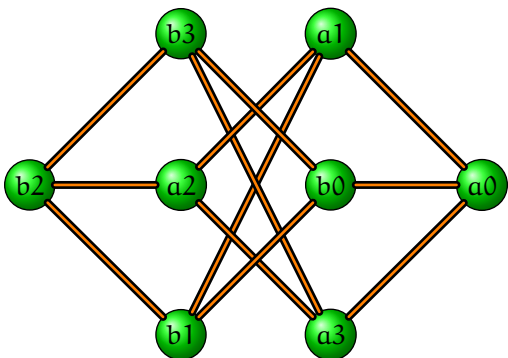
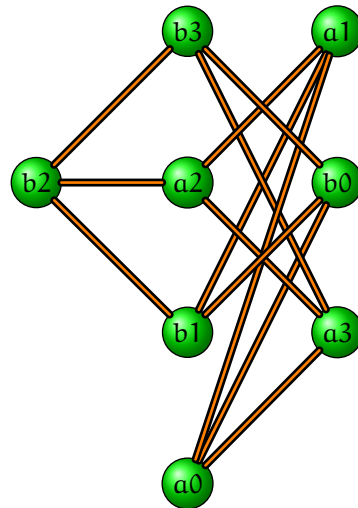
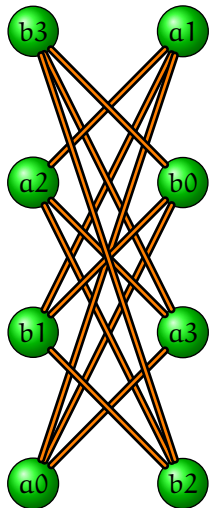
- Grafer kan brukes til å representere omtrent alt som fins av relasjoner.
- Mange algoritmer/egenskaper kan forstås bedre ved å bruke grafer.
- Ofte er løsningen å kunne identifisere et problem som et *grafteorisk* problem.

Grafisomorfier

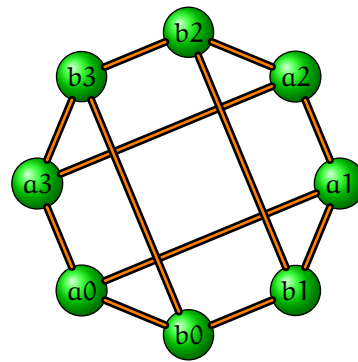
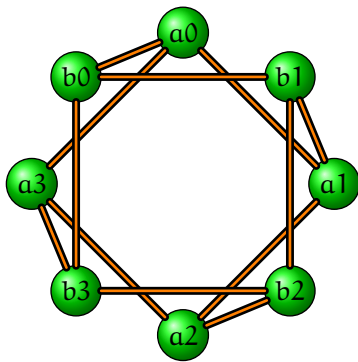
- Vi skal nå møte begrepet *isomorfi* for første gang i dette kurset.
- Isomorfibegrepet er veldig generelt og brukes overalt i matematikk.
- Ordet kommer fra gresk og betyr formlik (iso = lik, morf = form).
- Isomorfe objekter skal ha samme *form*, men kan ha ulikt innhold.
- En isomorfi er en funksjon som er injektiv og surjektiv – det er altså en en-til-en-korrespondanse – og som bevarer bestemte egenskaper.
- Intuitivt, så sier vi at to matematiske objekter er *isomorfe* hvis de er “strukturelt like”.
- Hvis vi har tegnet opp to grafer og kan komme fra den ene grafen til den andre ved kun å flytte på noder og endre på lengdene på kantene, så er grafene *isomorfe*.
- Vi har ikke lov til å legge til eller ta bort noder eller kanter, eller dele kanter eller noder i to...
- Følgende to grafer er isomorfe.



- Disse to grafene er også isomorfe med følgende grafer.



- De neste par grafene er også er isomorfe med disse.



- Vi skal nå gjøre isomorfibegrepet helt presist.
- Hvis G og H er to grafer, så er en isomorfi mellom grafene en funksjon fra nodene i G til nodene i H som oppfyller bestemte egenskaper.

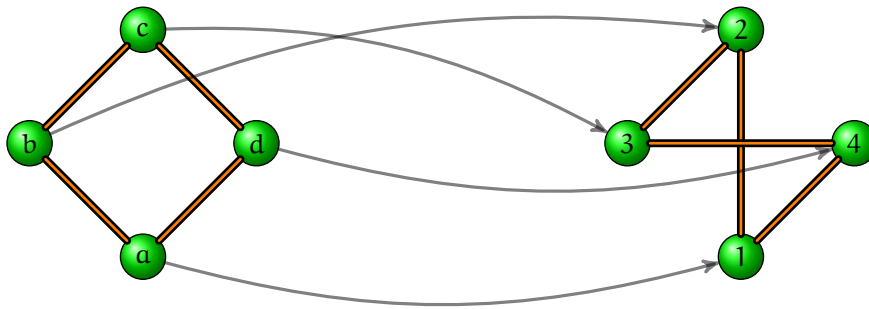
Definisjon (Isomorfi).

La G og H være to enkle grafer slik at $V(G)$ er mengden av noder i G og $V(H)$ er mengden av noder i H . En *isomorfi* fra G til H er en funksjon $f : V(G) \rightarrow V(H)$ med følgende egenskaper:

- f er surjektiv og injektiv
- Nodene u og v er naboer i G hvis og bare hvis nodene $f(u)$ og $f(v)$ er naboer i H .

Eksempel.

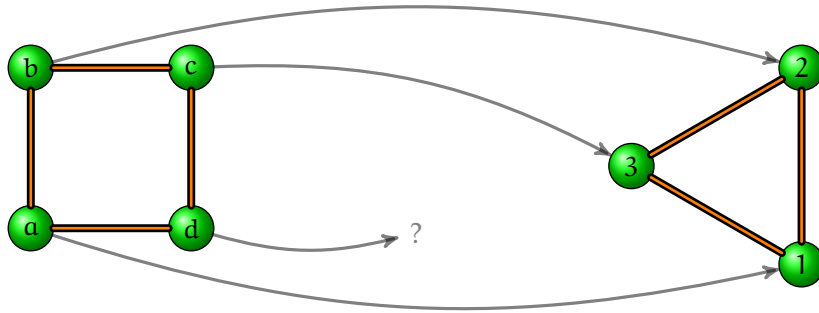
- La grafen G bestå av nodene $\{a, b, c, d\}$ og kantene $\{ab, bc, cd, da\}$.
- La grafen H bestå av nodene $\{1, 2, 3, 4\}$ og kantene $\{14, 34, 12, 32\}$.
- Funksjonen f slik at $f(a) = 1$, $f(b) = 2$, $f(c) = 3$ og $f(d) = 4$ er en isomorfi.
- F.eks. ser vi at a og b er naboer i G (siden ab er en kant).
- Da må $f(a)$ og $f(b)$, som er 1 og 2, være naboer i H .
- Det stemmer, siden 12 er en kant i H .



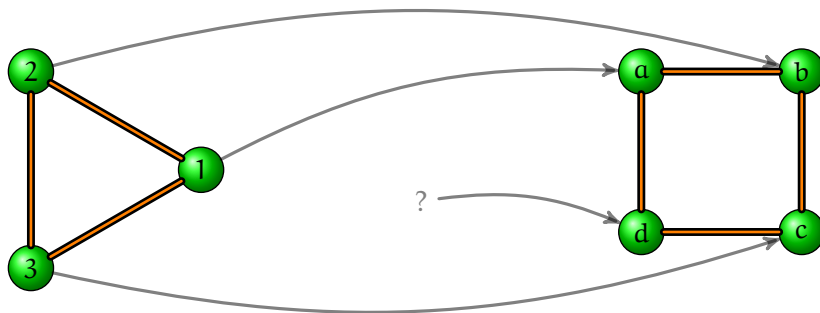
Merk.

- Det står "*hvis og bare hvis*" i definisjonen:
- Nodene u og v er naboer i G *hvis og bare hvis* nodene $f(u)$ og $f(v)$ er naboer i H .
- *Hvis*-delen av definisjonen er denne påstanden:
 - Nodene u og v er naboer i G *hvis* nodene $f(u)$ og $f(v)$ er naboer i H .
 - Det betyr at hvis nodene u og v *ikke* er naboer, så er heller ikke nodene $f(u)$ og $f(v)$ naboer.
 - "Naboer i $H \rightarrow$ Naboer i G "
- *Bare hvis*-delen av definisjonen er denne påstanden:
 - Nodene u og v er naboer *bare hvis* nodene $f(u)$ og $f(v)$ naboer.
 - Det betyr at hvis nodene u og v er naboer, så er også nodene $f(u)$ og $f(v)$ naboer.
 - "Naboer i $G \rightarrow$ Naboer i H "

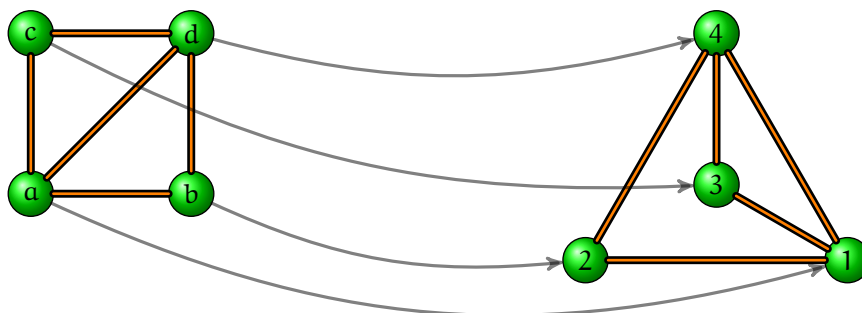
- For å vise at to grafer er isomorfe, må man gi en funksjon og argumentere for at funksjonen har egenskapene som er nødvendige.
- Det fins per i dag ingen effektiv algoritme for å avgjøre om to grafer er isomorfe.
- For å vise at to grafer *ikke* er isomorfe, så er det tilstrekkelig å finne en "grafteorisk egenskap" som kun den ene av grafene har.
- En grafteorisk egenskap er en egenskap som bevares under "lovlige" transformasjoner, som å flytte rundt på nodene, gjøre kantene lengre/kortere, etc.
- Noen av de enkleste grafteoriske egenskapene er f.eks.:
 - Hvor mange noder en graf har.
 - Hvor mange kanter en graf har.
 - Hvor mange noder av en bestemt grad en graf har.
 - Korteste avstand mellom to noder.
- Er følgende grafer isomorfe?



- Nei, de er ikke isomorfe.
- Grafen til høyre har *færre* noder enn grafen til venstre, så ingen funksjon fra den venstre grafen til den høyre kan være *injektiv* eller *en-til-en*.
- Er følgende grafer isomorfe?

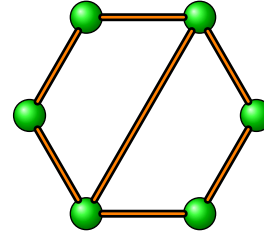
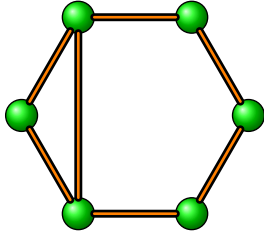


- Nei, de er ikke isomorfe.
- Grafen til høyre har *flere* noder enn grafen til venstre, så ingen funksjon fra den venstre grafen til den høyre kan være *surjektiv* eller *på*.
- Er følgende grafer, G og H, isomorfe?



- Ja, de er isomorfe.
- Funksjonen $f : V(G) \rightarrow V(H)$ gitt ved $f(a) = 1$, $f(b) = 2$, $f(c) = 3$ og $f(d) = 4$ er en isomorfi.

- Vi ser at u og v er naboer i G hvis og bare hvis $f(u)$ og $f(v)$ er naboer i H .
- Er følgende grafer isomorfe?



- Nei, de er ikke isomorfe.
- Grafen til venstre innholder tre noder som alle er relatert til hverandre; det gjør ikke grafen til høyre.

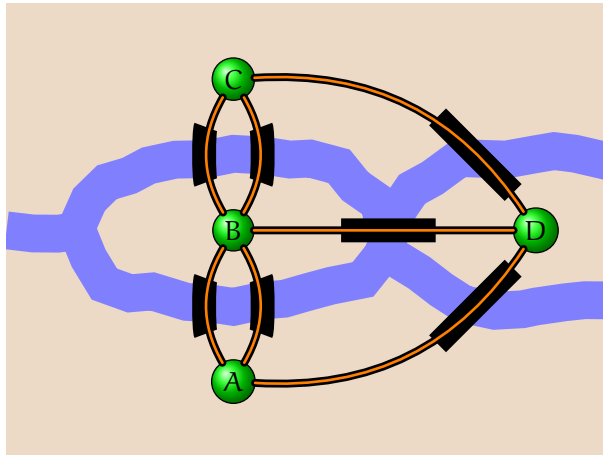
Noen kommentarer

- Å avgjøre om to grafer er isomorfe er i bunn og grunn å finne ut om det er den samme grafen man har å gjøre med.
- Hvis en rekke grafer er gitt, så ønsker vi å finne ut om noen av dem er isomorfe for å unngå å gjøre overflødig arbeid.
- Det er ingen som har klart å lage en *effektiv* algoritme (polynomiell tid) for å avgjøre om grafer er isomorfe (i det generelle tilfellet).
- Mange spesialtilfeller, f.eks. trær, vet man mye om.
- I praksis så klarer man å lage ganske effektive algoritmer allikevel, men i *verste* tilfelle må man backtrack over alle $n!$ mulige omdøpinger av nodene.
- Å finne isomorfier fra en graf til seg selv er en måte å avdekke *symmetrier* på.
- Å finne ut om en graf er en *delgraf* av en annen er et annet, men relatert, problem. (Ofte vanskeligere.)

Stier og kretser

- Vårt neste tema er *stier* (engelsk: *path*) og *kretser* (engelsk: *circuit*).
- Vi skal begynne med det klassiske eksemplet om *Königsbergs broer*.
- Kort fortalt har vi sju broer som forbinder fire landområder.
- Spørsmålet er om det går an å gå en tur i Königsberg slik at man går over hver av de sju broene *nøyaktig* én gang.
- Dette er kjent for å ha blitt løst av Leonhard Euler omkring 1735.
- Vi skal se at oppgaven er den samme som å finne en *Eulersti* i grafen som representerer Königsberg.

Königsbergs broer



Vi representerer situasjonen med en graf. Spørsmålet blir nå om det er mulig å gå over alle kantene nøyaktig en gang.

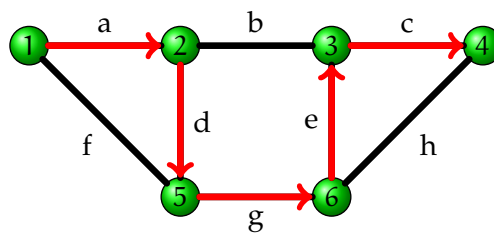
Definisjon (Sti).

En *sti* av lengde n i en graf er sekvens av noder og kanter på formen

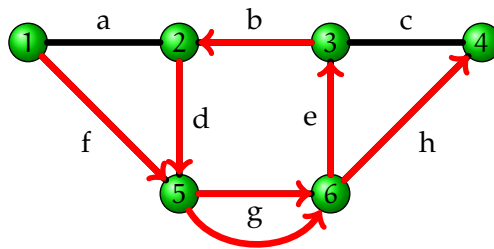
$$v_0 e_1 v_1 e_2 v_2 \dots e_n v_n$$

hvor e_i er en kant som forbinder v_{i-1} og v_i for $i \in \{1, 2, \dots, n\}$. En sti hvor $v_0 = v_n$ kalles en *krets*. Vi sier at sekvensen er en sti fra v_0 til v_n .

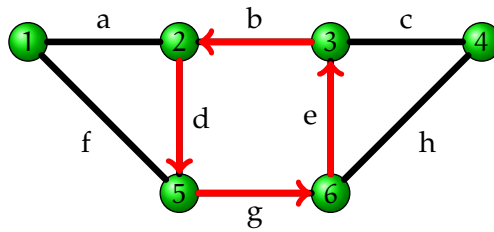
- En sti kalles også for en *vei*.
- *Lengden* til en sti er det samme som antall kanter i stien.
- En enkelt node er både en sti og en krets av lengde 0.
- Hvis grafen er *enkel*, tillater vi oss å skrive $v_0 v_1 v_2 \dots v_n$ for stien.
- Vær oppmerksom på at terminologien for grafteori varierer fra lærebok til lærebok.
- Vi ser på noen eksempler.



1a2d5g6e3c4 er en sti fra 1 til 4.

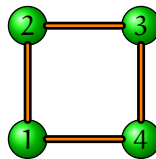


1f5g6e3b2d5g6h4 er en sti fra 1 til 4.



5g6e3b2d5 er en krets som begynner og slutter i 5.

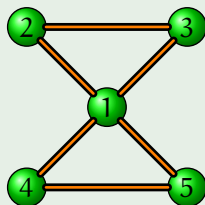
- Hvis en graf er enkel fins det ingen parallelle kanter, og da kan vi betegne en sti som en sekvens av noder.
- Det er vanlig å identifisere kretser som kun er forskjellige med hensyn på startnode eller rekkefølge.



- Her er 12341 en krets.
- Vi identifiserer denne med kretsene 23412, 34123, etc., og 43214, 32143, etc.

Oppgave.

Hvor mange *forskjellige* kretser som inneholder alle kantene nøyaktig én gang fins i følgende graf?



- Vi kan definere mengder av stier *induktivt* på følgende måte.

Definisjon (Mengden av stier - induktivt).

- En sekvens som består av en node v er en sti.
- Hvis p er en sti, og det siste elementet i p er noden u , og det går en kant fra u til v , så er sekvensen pev en sti.
- Mengden av stier er den minste mengden som oppfyller disse to kravene.

- Når vi nå har begrepet om en sti kan vi definere *sammenhengende grafer* mer presist.

Definisjon (Sammenghengende).

En graf er *sammenhengende* hvis det for hvert par av noder u og v fins en sti fra u til v .

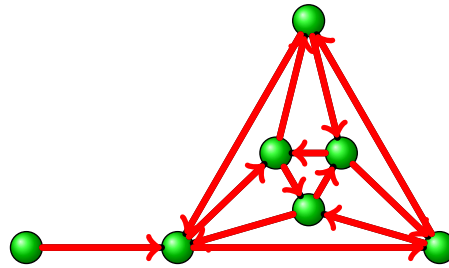
- Vi kan definere en ekvivalensrelasjon R på mengden av noder i en graf ved å si at uRv skal holde hvis det fins en sti fra u til v .
- Det er lett å sjekke at R er transitiv, refleksiv og symmetrisk.
(Vi tegner og forklarer på tavla.)
- Ekvivalensklassene definert av R kalles for *komponentene* til grafen.
- En graf kan deles opp i "sammenhengende delgrafer" på en slik måte.
- En sammenhengende graf er en graf som består av en komponent.

Definisjon (Eulersti/Eulerkrets).

La G være en sammenhengende graf. En *Eulersti* er en sti som inneholder hver kant fra G nøyaktig én gang. En *Eulerkrets* er en Eulersti hvor den første og den siste noden sammenfaller. En sammenhengende graf som har en Eulerkrets kalles *Eulersk*. En sammenhengende graf som har en Eulersti, men *ikke* en Eulerkrets, kalles *semi-Eulersk*.

- Noen kommentarer er på sin plass.
- I en Eulersti har vi lov til å gjenta noder, men ikke kanter.
- Eulerstier kalles også for *Eulerveier*.

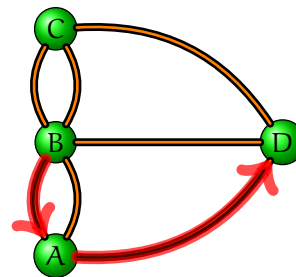
- Finner vi en Eulersti i denne grafen?
- Vi må i hvert fall begynne eller slutte i noden helt til venstre, siden den har grad 1.



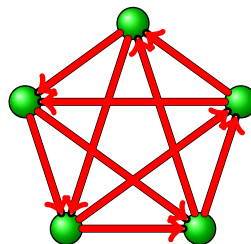
- Vi fant en Eulersti.
- En Eulerkrets er riktignok umulig, på grunn av noden til venstre.
- Ved å ta bort denne, så får vi en Eulerkrets.

Tilbake til Königsberg

- Problemet om Königsbergs broer blir nå: Har grafen over Königsberg en Eulersti?
- En sti må gå innom noden A minst en gang.
- Hvis vi går inn i A og ut igjen, så gjenstår én kant.
- For å gå over alle tre kantene som ligger inntil A, så må man enten begynne eller slutte i A.
- Det samme gjelder for C, og D og B, siden de alle har *odde grad*.



- Vi ser om vi finner en Eulerkrets i den komplette grafen med fem noder.



- Det var greit.
- Vi observerer at hver node har grad 4, et partall.
- Vi skal nå se at det er en sammenheng mellom eksistensen av Eulerstier/-kretser og hvorvidt gradene til nodene er partall.
- Anta at vi har en graf med en Eulerkrets.

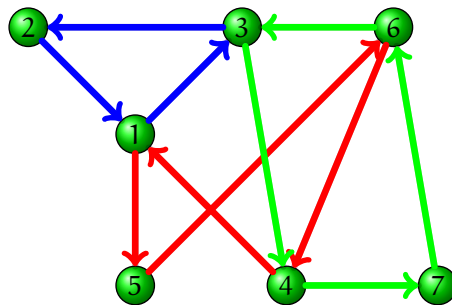
- Hva er da gradene til nodene i grafen?
- Vi kan skrive Eulerkretsen som sekvensen $v_0 e_1 v_1 e_2 v_2 \dots e_n v_n$, hvor $v_n = v_0$.
- Graden til en node u må være 2 ganger antall ganger den forekommer i sekvensen.
- Enhver node i grafen må da ha grad lik et partall.
- Vårt neste teorem sier det omvendte, nemlig at hvis hver node i en graf har grad som er et partall, så må grafen inneholde en Eulerkrets.

Teorem.

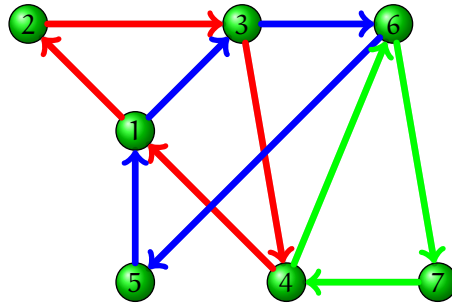
La G være en sammenhengende graf.

1. Hvis graden til enhver node i G er et partall, så inneholder G en Eulerkrets.
2. Hvis nøyaktig to noder i G har *odde* grad, så inneholder G en Eulersti som begynner i en node av odde grad og som slutter i en node av odde grad.
3. Hvis G har mer enn to noder av odde grad, så inneholder grafen *ikke* en Eulersti.

- Boka beviser påstand 1 ved å gi en algoritme for å konstruere en Eulerkrets for en graf hvor hver node har et partall som grad.
- Vi må da argumentere for at algoritmen er *korrekt*, at den alltid vil finne en Eulerkrets.
- I dette tilfellet er det ganske greit å se.
- Vi skal se at påstand 1 medfører påstand 2, at en graf som inneholder nøyaktig to noder av odde grad inneholder en Eulersti.
- La oss se på intuisjonen bak algoritmen.

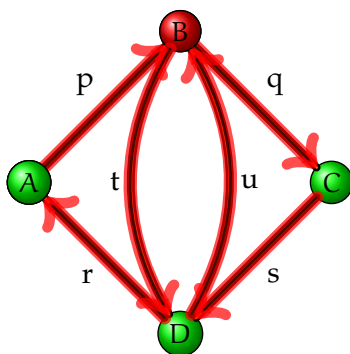


- Hvis vi begynner i node 2, så finner vi kretsen **2132**.
 - Siden ubrukte kanter ligger inntil både node 1 og 3, forsøker vi å *utvide* vår nåværende krets ved å legge til nye kretser.
 - Hvis vi begynner i node 1, så finner vi kretsen **15641**.
 - Vi kan *sette sammen* disse kretsene og få kretsen **21564132**.
 - Vi finner en krets til, **47634**.
 - Vi får da kretsen **215647634132** som er en Eulerkrets.
- Her er en annen måte å sette sammen flere kretser til en Eulerkrets på, med den samme grafen.



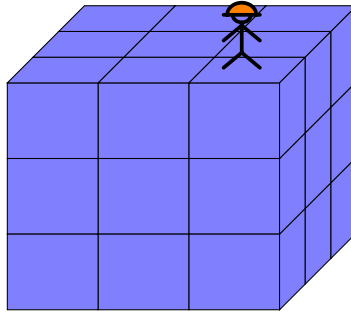
- Hvis vi begynner med en sti som kun inneholder én node og utvider stien stegvis ved å legge til kanter og stier, så må vi før eller siden komme tilbake til den første noden i stien, slik at vi får en *krets*.
- Grunnen er at hver node har grad som er et partall.
- Hver gang vi "går inn i" en node i stien, som ikke er startnoden, så må finnes en ubrukt kant slik at vi kan "gå ut" igjen.
- Før eller siden må vi komme tilbake til startnoden.

1. Input en Eulergraf G med noder V og kanter E
2. $krets \leftarrow$ en node fra V
3. **While** $E \neq \emptyset$ **do**
 - 3.1. $i \leftarrow$ den første noden i $krets$ med en kant fra E som ligger inntil i
 - 3.2. $v \leftarrow i$; $nykrets \leftarrow i$
 - 3.3. **Repeat**
 - 3.3.1. $e \leftarrow$ en kant fra E som ligger inntil v
 - 3.3.2. $v \leftarrow$ noden som er nabo med v via e
 - 3.3.3. $nykrets \leftarrow$ sammensetningen av $nykrets$ og e og v
 - 3.3.4. $E \leftarrow E - \{e\}$**until** ingen kant fra E ligger inntil v
 - 3.4. $krets \leftarrow$ sammensetningen av $krets$ før i , $nykrets$ og $krets$ etter i
4. Output $krets$



$E = \{p, q, s, r, t, u\}$
 $i = AB$
 $krets = ApBtDuBqCsDrA$
 $nykrets = ApBqCsDrABtDuB$

En nøtt om en maur og en kube



- Anta at en maur sitter på utsiden av en 3x3x3-kube.
- Mauren spiser seg inn i den første lille kuben (av i 27 kuber).
- Er det mulig for mauren å spise seg gjennom alle kubene og så komme ut igjen samme sted?
- Mauren har kun lov til å gå en kube av gangen og ikke på skrå.

Forelesning 23

Grafteori, Trær

Dag Normann - 16. april 2008

Oppsummering

- En graf består av *noder* og *kanter*
- Kanter ligger *inntil* noder, og noder kan være *naboer*.
- Vi bør kjenne til begrepene om *sammenhengende* grafer, *tomme* grafer, *løkker*, *parallelle* kanter, *enkle* grafer og *komplette* grafer.
- Hver node har en *grad*.
 - Summen av gradene til alle nodene i en graf er lik 2 ganger antallet kanter.
 - Håndhilselemmaet: Det er alltid et partall antall noder av odde grad i en graf.
- Vi skal kjenne til *komplementet* av en graf og *matriserepresentasjoner*.
- Grafer kan brukes til å representere omtrent alt som fins av relasjoner.
- Mange algoritmer/egenskaper kan forstås bedre ved å bruke grafer.
- Ofte er løsningen å kunne identifisere et problem som et *grafteorisk* problem.
- Vi har sett på isomorfibegrepet for grafer.
- To grafer er isomorfe hvis alle de viktige egenskapene er de samme.
- Mer presist
 - Det finnes en bijeksjon mellom nodene og mellom kantene slik at bildet av en kant går mellom bildet av to noder hvis og bare hvis kanten går mellom nodene.
- Vi definerte stier og
- kretser
- En sti er en følge av noder og kanter slik at vi går fra node til node via kantene mellom dem.

Den presise definisjonen ble gitt mandag.
- En krets er en sti som begynner og slutter samme sted.

Digresjon: Firefarveproblemet

- I mange, mange år var følgende et åpent matematisk problem:

*Anta at vi har et plant kart over landområder (land, fylker, stater o.l.).
Er det alltid mulig å trykke kartet ved hjelp av bare fire farver slik at to landområder som grenser opp mot hverandre alltid har forskjellig farve.*

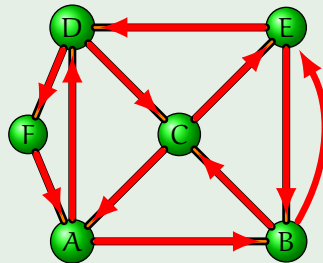
- Hvis vi representerer landene som noder og grensene som kanter, er dette egentlig et grafteoretisk problem.
- Grafteori, som en matematisk tung disiplin, har mye å hente fra forsøkene på å løse dette problemet.
- Måten problemet ble løst på har interesse i seg selv.
- De som løste det, reduserte problemet til et stort antall enkelttilfeller, som deretter ble sjekket av en datamaskin.
- Var det mennesker eller datamaskinen som løste problemet?

Eulerstier

- En Eulerkrets er en krets i en graf som inneholder hver kant nøyaktig en gang.
- Euler fant ut at en graf har en Eulerkrets nøyaktig når den er sammenhengende og hver node har et partall som grad.
- Mandag så vi på en algoritme som fant en Eulerkrets når det var mulig.
- Den kan også brukes til å finne en Eulersti, det vil si en sti som er innom alle kantene nøyaktig en gang, men som kan begynne og slutte på forskjellige steder.
- Disse stedene må da være de to nodene med odde grad.
- Utvider vi grafen med en kant mellom disse to nodene, kan vi lage en Eulerkrets.
- Tar vi bort den nye kanten, får vi en Eulersti.

Eksempel.

La oss se på et eksempel:



- Det er to strategier for å bevise en setning som Eulers.
- Vi kan bruke et induksjonsbevis.
- Induksjonsbevis gir ofte opphav til algoritmer, og den algoritmen boka presenterer kan sees på som utledet av et induksjonsbevis.
- Vi kan også bevise teoremet *direkte*, uten å argumentere via algoritmen.
- Her følger et bevis for at det er alltid finnes en Eulerkrets hvis hver node har grad lik et partall.
- Det er også mulig å trekke en algoritme ut av dette beviset, og algoritmen blir veldig lik den vi så på mandag.

Bevis (1).

La G være en sammenhengende graf hvor gradene til alle nodene er partall. La S være den lengste stien

$$v_0 e_1 v_1 e_2 v_2 \dots e_n v_n$$

slik at ingen kant forekommer to ganger. Vi skal vise at S er en Eulerkrets. Vi skal gjøre dette ved å vise følgende tre påstander:

- (a) S er en krets.
- (b) S inneholder alle nodene i grafen.
- (c) S inneholder alle kantene i grafen.

Siden ingen kant forekommer to ganger, må S være en Eulerkrets.

Bevis (Fortsatt).

(a) Noden v_0 må være lik v_n . Når vi går ut av den første noden, v_0 , via kanten e_1 , så bruker vi opp én av kantene som ligger inntil v_0 . For hver node vi går inn i og ut av, så bruker vi opp to kanter. Når vi er fremme ved den siste noden i stien, v_n , så fins det ingen ubrukt kant som ligger inntil v_n . Hadde det vært en slik kant, så ville vi hatt en sti som var lenger enn S , og da hadde ikke S vært maksimal. Siden graden til v_n er et partall, så må vi tidligere i stien ha gått ut av v_n . Den eneste muligheten er at v_n er lik v_0 . Dermed er S en *krets*.

Bevis.

(b) S må bestå av alle nodene i grafen. Det er fordi grafen er sammenhengende og S er maksimal. Hvis en node v ikke hadde vært med, så kunne vi ha laget en krets som var lenger enn S .

Bevis.

(c) S inneholder alle kantene fra grafen. Anta for motsigelse at det fins en kant e , som forbin-der nodene u og v , som ikke er med i S . Siden S inneholder alle nodene fra grafen, så må v være lik v_k for en passende k . Da kan vi lage en sti som er lenger enn S ved å begynne med ue og fortsette med S :

$$ue v_k e_{k+1} v_{k+1} \dots e_n v_n \underbrace{e_1 v_1 e_2 v_2 \dots e_{k-1} v_{k-1} e_k v_k}_S$$

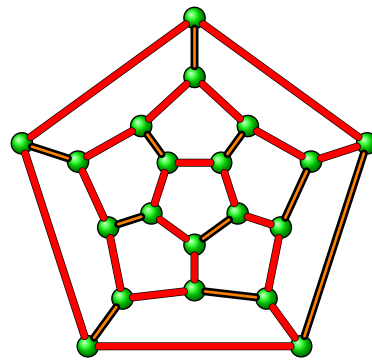
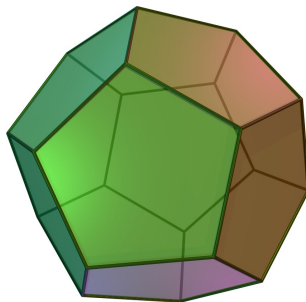
Hamiltonstier

- Vi må også si litt om stier som inneholder alle *nodene* i en graf, uavhengig hvorvidt alle kantene er med eller en kant er med flere ganger.
- “Den handelsreisendes problem” er et slikt problem, hvor man er ute etter den *korteste* stien som går gjennom alle byene i en mengde.

Definisjon.

La G være en sammenhengende graf. En *Hamiltonsti* er en sti som inneholder hver node fra G nøyaktig én gang. En *Hamiltonkrets* er en Hamiltonsti hvor den første og den siste noden sammenfaller. En sammenhengende graf som har en Hamiltonkrets kalles *Hamiltonsk*.

- Hamiltons puzzle tar utgangspunkt i et dodekaeder (et av de fem Platonske legemene) hvor hvert hjørne er merket med navnet på en by. Spørsmålet han stilte var om det var mulig å reise gjennom alle byene nøyaktig én gang. Vi ser at dette spørsmålet er det samme som om den tilhørende grafen har en Hamiltonsti.



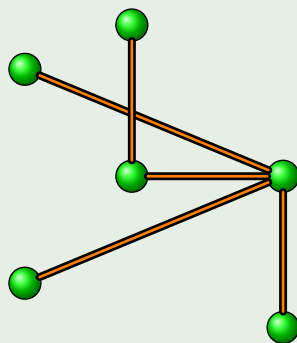
- Euler studerte også et tilsvarende problem: når det er mulig for en springer å gå over *alle* rutene på sjakkbrett av ulike størrelser.
- Det er heller ingen som har klart å lage en *effektiv* algoritme for å finne ut om det fins en Hamiltonkrets i en graf.
- Dette er “like vanskelig” som å bestemme om et utsagnslogisk utsagn er en tautologi. [Det tilhører klassen av **NP**-komplette problemer.]
- I praksis er det sjeldent at man virkelig trenger å finne en Hamiltonkrets.
- Ofte er det tilstrekkelig å finne en Eulerkrets, eller greit å gå over noder flere ganger.
- Det fins mange spesialtilfeller og heuristikker man kan benytte seg av.

Trær

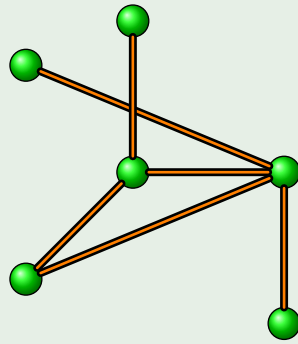
- Et tre er en spesiell type graf.

- Intuitivt er et tre noe som vokser fra en rot og så forgrener seg uten noe sted å vokse sammen igjen.
 - Vi kan se på et biologisk tre som en graf, ved å la hvert forgreningspunkt være nodene, og delene av en stamme, gren eller kvist mellom to forgreningspunkter være kantene.
 - Vi skal gi en presis definisjon av når en graf kan betraktes som et tre.
-
- Hvorfor skal vi lære om trær?
 - Hvis en graf representerer et nettverk, vil et tre svare til et nettverk hvor det bare finnes en sti fra en node til en annen.
 - Hvis nettverket består av kabler eller andre medier som formidler informasjon, kan det være hensiktsmessig at signaler bare går langs en vei, slik at systemet ikke forstyrres av at samme informasjon kommer med små tidsforskjeller.
 - Dataobjekter som sammensatte algebraiske uttrykk, utsagnslogiske formler eller program har ofte en trestruktur som beskriver hvordan komplekse objekter er bygget opp fra enklere objekter.
 - For å undersøke om et utsagn formalisert i matematikken kan bevises eller ikke, kan man prøve å bygge opp et tre av utsagn hvor forgreningen stopper når vi har nådd aksiomene.
- Denne naive ideen danner grunnlag for enkelte automatiske bevis-søkere.

Eksempel.



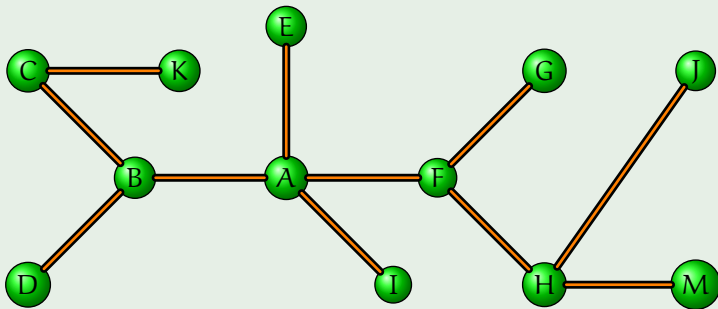
Eksempel.



Definisjon.

- a) En sykel (Eng. cycle) i en graf er en sti med følgende egenskaper:
- Stien inneholder minst en kant.
 - Ingen kant forekommer mer enn en gang.
 - Stien er en krets, dvs. den begynner og slutter i samme node.
- b) En graf er et tre hvis grafen er *sammenhengende* og grafen ikke inneholder noen sykler.

Eksempel.



Vi ser at denne grafen har 12 noder og 11 kanter.

Eksempel.

Et tre trenger ikke å ha noen forgreningspunkter:



Her har vi fem noder og fire kanter.

- I de eksemplene vi har sett på har vi alltid endenoder i et tre, det vil si noder av grad 1.
- Husk at en graf alltid har minst en node.

Grafen med en node og ingen kanter er et tre. Alle andre trær vil ha endenoder.

- I de eksemplene vi har sett har alle trærne en node mer enn de har kanter.

Dette er en egenskap som alle endelige trær har.

Det er ingenting i definisjonen av grafer og trær som sier at de skal være endelige, men vi kommer til å begrense oss til endelige grafer og trær hvis vi ikke sier noe annet. Boka forutsetter også at vi bare arbeider med endelige grafer og trær i dette kurset

Teorem.

- a) Hvis et tre har minst en kant, har treet en node med grad 1 (En slik node kaller vi en endenode eller bladnode).
- b) I ethvert tre finnes det nøyaktig en node mer enn det finnes kanter.

Bevis.

- a) La

$$v_0 e_1 \cdots e_n v_n$$

være en sti med maksimal lengde.

Siden grafen er et tre, kan ikke stien være innom samme node to ganger.

Endenodene v_0 og v_n må være bladnoder, siden vi ellers ville kunnet gjøre stien lengere.

Bevis (Fortsatt).

- b) Vi bruker induksjon på antall noder i treet.

Hvis det bare finnes en node, har vi ingen kanter, og påstanden stemmer.

Hvis det finnes mer enn en node, kan vi anta at påstanden holder for alle mindre trær.

Tar vi bort en bladnode og den ene kanten som ligger inntil denne noden, får vi et mindre tre.

Siden vi har tatt bort en node og en kant, og da har en node mer enn vi har kanter, må dette være tilfellet i det gitte treet også.

Resonnementet illustreres på tavla.

Vektete grafer

- Hvis en graf representerer et veinett er det av interesse å vite hvor lange de enkelte veistrekningene er.

- Hvis en graf representerer et ledningsnett, kan anleggskostnader og driftskostnader ved de enkelte strekningene være av interesse.
- Hvis nodene i en graf står for land og kantene for grenseoverganger mellom dem, kan tollsatsene eller korrupsjonskoeffisienten ved de forskjellige grenseovergangene bety noe.
- Siden vi har mange eksempler på grafer hvor det er viktige tallstørrelser knyttet til de enkelte kantene, studerer vi vektete grafer som et eget begrep.

Definisjon.

En vektet graf er en graf hvor hver kant har fått en vekt, et positivt reelt tall.

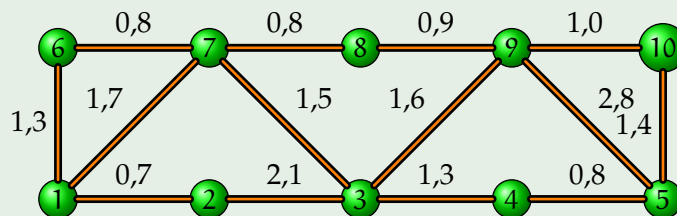
Merk.

Formelt sett kan vi definere en vektet graf som et par (G, f) hvor G er en graf og f er en funksjon fra mengden av kanter i G til de positive reelle tallene.

Vi har altså bruk både for ordnede par og for funksjoner for å gi en skikkelig definisjon.

Eksempel.

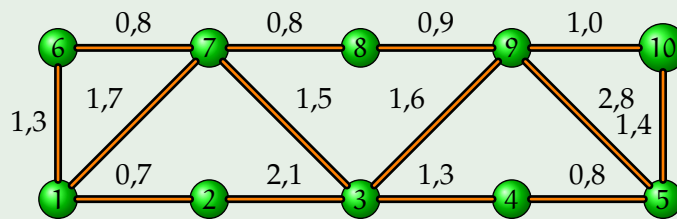
La oss se på et eksempel:



- Det er mulig å trekke kabler mellom disse skjematisk tegnede byene, hvor kostnaden målt i antall NOK 10^7 :

Kan vi fjerne noen av kantene slik at anleggskostnadene blir minst mulig, men vi fortsatt forbinder alle byer med kabler?

Eksempel (Fortsatt).



Så lenge grafen inneholder kretser, må det være greit å ta bort en kant i kretsen. Vi bør derfor finne det mest kostnadseffektive deltreet som når over alle nodene. Vi skal komme tilbake til dette eksemplet når vi har diskutert algoritmen som ligger bak.

Definisjon.

La G være en sammenhengende graf, og la T være et deltre av G . Det betyr her at T og G har de samme nodene, alle kantene i T er kanter i G , men noen kanter i G kan mangle i T .

Vi sier at T spenner ut G hvis alle nodene i G ligger inntil en kant i T . (Tegning på tavla).

Husk at et tre er en sammenhengende graf, så dette betyr at alle par av forskjellige noder i G kan forbindes med en (og bare en) sti i T .

- Hvis G er et vektet tre, er problemet å finne et tre T som spenner ut G slik at summen av vektene på kantene i T blir minst mulig.

- Prims algoritme gir en metode for å finne det minimale utspennende treet til en vektet graf.
- I læreboka står det en pseudokode for Prims algoritme.
- Her vil vi beskrive algoritmen litt mer uformelt.
- Det viser seg at hvis man bygger opp et tre ved i hvert skritt å gjøre det som i øyeblikket virker mest fornuftig, så kommer man frem.
- Korrekthetsbeviset for Prims algoritme skal vi ikke legge vekt på, men det forventes at man kan praktisere den på eksempler.
- Vi beskriver Prims algoritme litt anderledes enn den er formulert i læreboka, men effekten er den samme, vi får det samme treet bygget opp i den samme rekkefølgen.

Forelesning 24

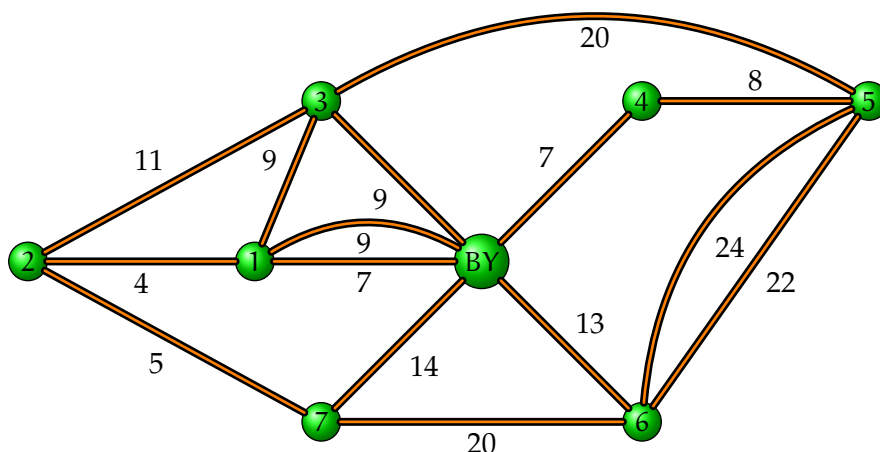
Trær

Dag Normann - 21. april 2008

Vektete grafer

- Vi har snakket om grafer og trær. Av begreper vi så på var
 - Eulerkretser og Eulerstier
 - Hamiltonkretser
 - Vektete grafer
 - Minimale utspennende trær.
- Vi skal nå se på et realistisk eksempel på en situasjon som langt på vei kan modelleres som en vektet graf, og hvor det vil være relevant å finne en Eulerkrets eller sti, en Hamiltonkrets og et minimalt utspennende tre for å løse visse samfunnsoppgaver.
- I virkelighetens verden finner man ofte ikke en Eulersti når man trenger en eller en Hamiltonkrets når man trenger en, men som vi skal se, kan man alltid finne minimale utspennende trær.
- Vårt eksempel er en graf som modellerer veinettet mellom de lokale tettstedene i en kommune, og vektingen av kantene er antall kilometer hver enkelt veistrekning er på. Grafen er ikke *enkel*, men bortsett fra det er den som en vektet graf.

En kommunegraf



- Snøbrøyterne: Finnes det en Eulersti?
- Postutkjørerne: Finnes det en Hamiltonkrets?
- Bredbåndutbyggerne: Finnes det et minimalt utspennende tre?

En kommunegraf

Oppgave.

- a) Avgjør om det finnes en Eulerkrets eller en Eulersti i kommunegrafen, og finn i så fall denne.
- b) Er spørsmålet om det finnes en Hamiltonkrets det rette spørsmålet? Kunne postutkjørerne stilt et mer fornuftig grafteoretisk spørsmål?
- c) Finn et minimalt utspennende tre (bruker stoff fra resten av forelesningen).
For å få en vektet graf i tråd med definisjonen, kan du ta bort unødige kanter med mye vekt der det finnes parallelle kanter.

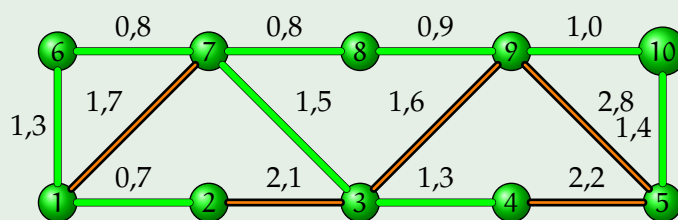
- Vi minner om at en *vektet graf* er en enkel graf hvor hver kant har en vekt, et ikke-negativt reelt tall.
- Hvis G er en sammenhengende graf, vil et utspennende tre være en delgraf T slik at
 - T har de samme nodene som G .
 - T er et tre, det vil si, T er sammenhengende og inneholder ingen sykler.
- Vi så på onsdag at hvis G har n noder, vil et utspennende tre ha $n - 1$ kanter.
- En konsekvens er at hver gang vi velger ut $n - 1$ kanter fra G slik at vi ikke får noen sykler, så får vi et utspennende tre.
- Det kan finnes mange forskjellige utspennende trær i en graf.
- Hvert slikt tre vil ha en samlet vekt, ved at vi legger sammen vektene på kantene.
- I en situasjon hvor vektene representerer kostnader, og hvor det er teknologisk nødvendig eller tilstrekkelig å erstatte grafen med et utspennende tre, er det av interesse å kunne finne et utspennende tre med minst mulig vekt.
- Det finnes effektive algoritmer for å kunne gjøre dette.
- Vi skal se på en slik algoritme.
- Prims algoritme gir en metode for å finne det minimale utspennende treet til en vektet graf.
- I læreboka står det en pseudokode for Prims algoritme.
- Her vil vi beskrive algoritmen litt mer uformelt.
- Det viser seg at hvis man bygger opp et tre ved i hvert skritt å gjøre det som i øyeblikket virker mest fornuftig, så kommer man frem.
- Vi skal ikke gi et korrekthetsbevis for Prims algoritme, men det forventes at man kan praktisere den på eksempler.
- Vi beskriver Prims algoritme litt anderledes enn den er formulert i læreboka, men effekten er den samme, vi får det samme treet bygget opp i den samme rekkefølgen.
- La G være en vektet, sammenhengende graf med noder $V = \{v_1, \dots, v_n\}$.
- La T_1 være treet som består av v_1 og ingen kanter.

- Start med node v_1 og la $V_1 = \{v_2, \dots, v_n\}$, altså resten av nodene.
- Finn $v_{i_1} \in V_1$ med minimal avstand til v_1 via kant e_1 .
- Vi får V_2 ved å fjerne v_{i_1} fra V_1 og vi får T_2 ved å legge til v_{i_1} og e_1 til T_1 .

Fortsettelse av algoritmen

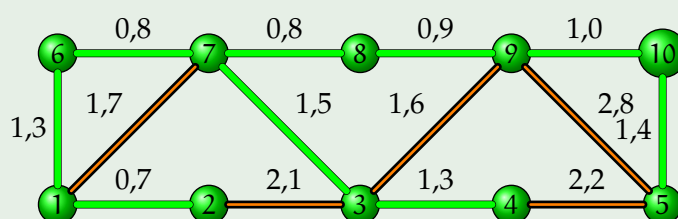
- Deretter fortsetter vi med alltid å velge den ubrukte noden som ligger nærmest, via en kant, til treet bygget opp så langt, og vi bygger ut treet med denne noden og den tilsvarende kanten.
- Siden grafen er sammenhengende, vil vi alltid finne en ny node som er "nabo" til treet bygget opp på et gitt tidspunkt, og da finner vi alltid en ny node som ligger nærmest.
- Vi skal illustrere hvordan denne algoritmen virker på eksemplet vi har gitt på en vektet graf.
- Forklaringen på hvorfor hvert enkelt skritt blir som det blir, gis muntlig på forelesningen.

Eksempel (Fortsatt).



- Vi skal følge hvordan Prims algoritme vil bygge opp et utspennende tre med minimal vektning skritt for skritt.
- Vi starter i Node 1:

Eksempel (Fortsatt).



- Så lenge vi har listet opp kantene i rekkefølge og alltid velger den kanten med minst vekt som kommer først i listen vår, spiller det ingen rolle hvor vi starter.

- Hvis vi starter i Node 8 bygger vi opp treet slik:
Det ble det samme treet til slutt.

Vi gir en pseudokode for Prim's algoritme.

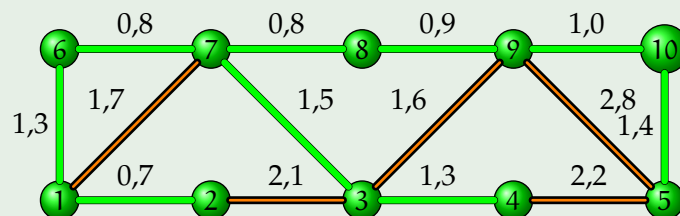
Den ser litt anderledes ut enn den som står i boka, men effekten skritt for skritt er den samme.

Detaljene står på neste side:

- 1 *Input* $V = \{v_1, \dots, v_n\}$ [Nodene i G]
- 2 *Input* $E = \{e_1, \dots, e_k\}$ [Kantene i G]
- 3 $T \leftarrow \{v_1\}$
- 4 $K \leftarrow \emptyset$
- 5 **While** $E \neq \emptyset$ **do**
 - 5.1 $x \leftarrow$ første $e_i \in E$ slik at e_i ligger inntil en node i T og har minimal vekt blant disse.
 - 5.2 $y \leftarrow$ noden ved x som ikke ligger i T
 - 5.3 $K \leftarrow K \cup \{x\}$
 - 5.4 $T \leftarrow T \cup \{y\}$
 - 5.5 $E \leftarrow E - \{e_i ; e_i \text{ ligger inntil to noder i } T\}$.
- 6 *Output* (T, K) .

- Det finnes en fremgangsmåte som ikke er avhengig av at vi starter noe sted, og som også vil gi oss det samme treet.
- Her bygger vi ikke opp et tre, men små deltrær som til sist vil vokse seg sammen til et tre.
- Vi utnytter at vi får et utspennende tre bare vi tar med $n - 1$ kanter uten å lage noen sykler.
- Hver gang legger vi til en ny kant med minimal vekt slik at vi ikke får noen sykler.
- Finnes det flere aktuelle kanter med samme minimale vekt, velger vi den som står først i listen vår.
- I eksemplet vårt vil da det utspennende treet bygge seg opp slik som på neste side.

Eksempel (Fortsatt).



Vi får fortsatt det samme treet

- Et annet naturlig problem i forbindelse med vektete grafer er å finne et utspennende tre slik at hver node har en minimal avstand til en gitt "sentralnode".
- Her tenker vi oss at vektene svarer til lengder av de enkelte kantene.
- Det finnes effektive algoritmer for dette også.
- Vi skal gi en uformell beskrivelse av Dijkstras algoritme og vise hvordan den virker på eksemplet vårt.
- Vi skal se at vi denne gangen får et annet tre.
- For dette problemet er også treet vi får avhengig av hvilken node som velges som "sentrum".

Anta at vi har en vektet graf G med en opplisting av n noder og endel kanter.

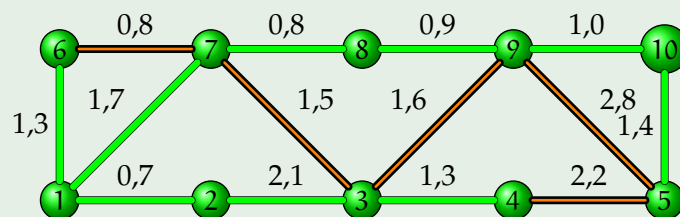
Vi starter med en *sentrumsnode* v_1 og lar T_1 bestå av noden v_1 og ingen kanter.

Ved rekursjon på $i \leq n$ konstruerer vi et tre T_i med i noder og $i - 1$ kanter fra G .

Vi konstruerer T_{i+1} fra T_i når $i < n$ ved følgende prosedyre:

1. Finn den noden v utenom T_i og den kanten e som er slik at e forbinder v til T_i , og vi oppnår den kortest mulige veien fra en ny node til v_1 via e og T_i på denne måten.
 2. Finnes det flere like gode alternativer, velg den v 'en og deretter den e 'en som står først i listene.
 3. Utvid T_i til T_{i+1} ved å legge til noden v og kanten e .
- Dijkstras algoritme er beskrevet i form av en pseudokode i boka.
 - Det er meningen at dere skal kunne finne det utspennende treet som gir de korteste stiene fra provinsen til sentrum når dere har fått gitt en vektet, sammenhengende graf.
 - La oss se på eksemplet vårt en gang til.

Eksempel.



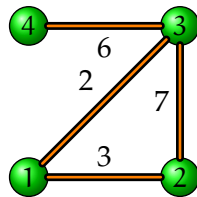
Vi starter i Node 1

Vi får et nytt tre.

- Når vi beskriver algoritmer som finner bestemte typer trær i vektete grafer, ligger det selvfølgelig under at vi tenker oss at disse algoritmene skal kunne programmeres.
- Det betyr at det må være mulig å representere disse vektete grafene digitalt.

- Vi har tidligere sett hvordan grafer kan representeres som matriser.
- Siden matriser kan representeres digitalt, betyr det at også grafer kan representeres digitalt.
- Vektete grafer kan også representeres som matriser.
- Vi følger boka, og lar ∞ representere at det ikke finnes noen kant mellom to noder.
- Hvis vi tolker grafen som en strømkrets hvor vektene representerer motstanden i hver enkelt ledning, vil det at vi ikke har noen direkte kobling mellom to noder svare til at vi har en ledning med uendelig motstand mellom dem.
- Vi illustrerer dette med ett eksempel.

En vektet graf:

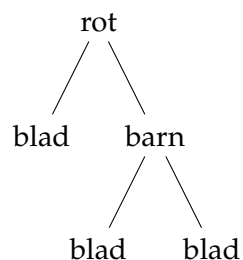


Matriserepresentasjonen:

	1	2	3	4
1	0	3	2	∞
2	3	0	7	∞
3	2	7	0	6
4	∞	∞	6	0

Trær med rot

- Til nå har vi sett på trær som sammenhengende grafer uten løkker.
- Det betyr at vi ikke har noe dynamisk bilde av disse trærne, de har ikke noe startpunkt eller noen rot.
- For mange anvendelser av teorien for trær, er det nyttig å kunne betrakte den ene noden som roten til treet, den noden alt annet har vokst ut fra.
- Formelt sett er et tre med rot definert som et grafteoretisk tre hvor en av nodene er utpekt som rot.
- Det er imidlertid vanlig å tegne slike trær litt anderledes enn trær uten rot.

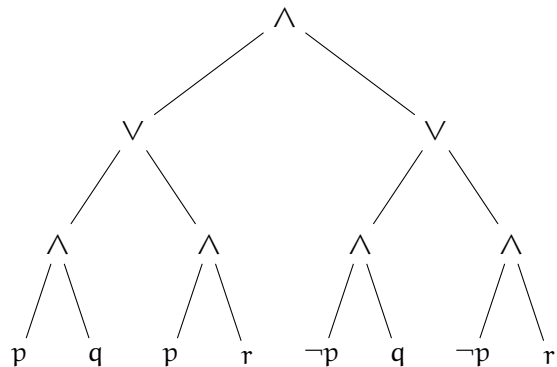


- Roten tegnes øverst, og treet "vokser" nedover.

- Nodene ligger i “lag” avhengig av avstanden til roten.
- Vi kan snakke om barna til en node, og om bladene til et tre med rot.

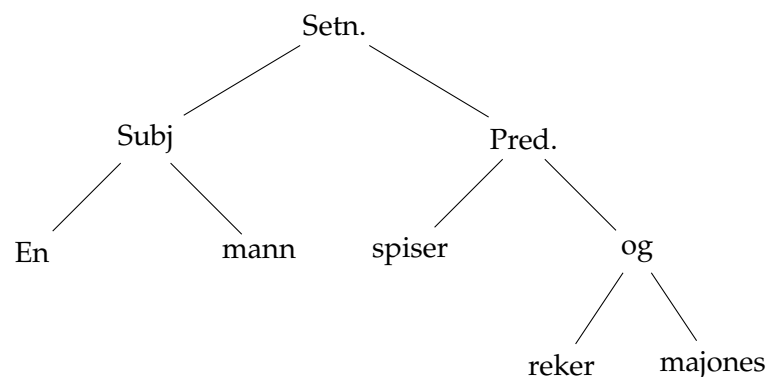
Vi skal se på endel eksempler før vi går nærmere inn på terminologien.

- Vi kan bruke trær til å gi en grafisk fremstilling av en utsagnslogisk formel.
- La $A = ((p \wedge q) \vee (p \wedge r)) \wedge ((\neg p \wedge q) \vee (\neg p \wedge r))$.
- Mengden av formler er bygget opp induktivt, og oppbyggingen av hver formel kan beskrives som et tre:



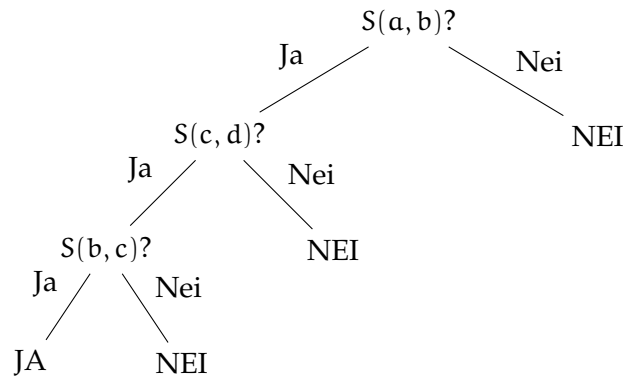
- Et tre som det på forrige side kaller vi ofte et syntakstre.
- Et syntakstre forteller oss hvordan et ord i et litt komplisert formelt språk er bygget opp av enklere ord.
- Syntakstrær kan brukes i grammatikk-analyse av setninger i naturlige språk.
- Da setter man opp et tre som beskriver hvordan en setning er bygget opp av subjekt og predikat, hvordan subjektet kan bestå av en artikkel og et substantiv osv.
- Som et eksempel skal vi se et slikt tre på neste side, uten at bruken av slike trær skal være noe tema for disse forelesningene:

- *En mann spiser reker og majones*
- Denne setningen er bygget opp slik:



- Trær kan ofte brukes til å beskrive forskjellige algoritmer hvor man stiller visse spørsmål, og prosessen videre avhenger av svarene på de enkelte spørsmålene.

- I læreboka står det et eksempel på et tre av spørsmål som kan brukes til å bestemme rekkefølgen på tre forskjellige tall a , b og c .
- Vi skal se på en spørsmålsserie som avgjør om fire individer a, b, c, d tilhører samme art.
- Vi skriver $S(x, y)$ for at x og y tilhører samme art.
- Det å tilhøre samme art er en ekvivalensrelasjon, og korrekthet av programmet bygger bare på det faktum (på samme måte som korrekthet av eksemplet i boka bygger på at vi har en total ordning).



- Det er verd å merke seg at prosedyren over faktisk sjekker om a, b, c og d står i relasjon S til hverandre når S er en transitiv relasjon.
- Vi sjekker først $S(a, b)$.
- Får negativt svar gir algoritmen negativt svar.
- Får vi positivt svar sjekker vi $S(c, d)$.
- Får vi positivt svar her også sjekker vi til slutt $S(b, c)$.
- Hvis S er transitiv vet vi at a, b, c og d ligger etterhverandre i S -betydningen.
- En algoritme som virker for veldig mange tolkninger av input kalles gjerne en polymorf algoritme, det vil si at den virker på mange strukturer.
- Sorteringsalgoritmer er eksempler på polymorfe algoritmer.

Forelesning 25

Trær

Dag Normann - 23. april 2008

Beskjeder

- Roger har bedt meg gi følgende beskjeder:
 - 1 Det meste av plenumsregningen i morgen, 24/4, blir tavleregning, slik at studentene ikke kan belage seg på finne alt på foiler senere.
 - 2 Roger vil repetere induksjonsbevis i forskjellige varianter, etter ønske fra mange studenter.
- Neste uke vil vi forelese stoff som går ut over boka, men som vil bli betraktet som pensum.
- Vi vil gå mye mer inn på bruk av trær i informatikk-sammenheng enn det boka gjør, og kan etterspørre noen teknikker vi beskriver til eksamen.
- Vi vil spesielt se på det som kalles unifiseringsalgoritmer.

Torsdag 1. mai er det selvfølgelig ingen plenumsregning.

Oppsummering

- Mandag brukte vi mye tid på å illustrere Prims algoritme.
- Prims algoritme vil finne det minste utspennende treet i en vektet graf.
- Prims algoritme er "grådig" i den forstand at den vurderer lokalt hva som er det beste neste skrittet.
- Vi gir ikke korrekthetsbeviset for Prims algoritme, men nevner at resultatet blir det samme uansett hvor i grafen man begynner.
- Prims algoritme er svært eksamensrelevant.
- Vi ga også en kort innføring i Dijkstras algoritme for å finne "korteste vei"-treet i en vektet graf.
- Da utpeker vi en av nodene i grafen som "sentrum", og tar sikte på å finne det treet som gir kortest mulig vei fra hver av de andre nodene til sentrum.
- Poenget med Dijkstras algoritme er at man hver gang legger en ny kant til en ny node til treet slik at den nye veistrekningen fra en ny node til sentrum blir kortest mulig.
- Vi illustrerte Dijkstras algoritme med et eksempel.
- Dijkstras algoritme brukes også til å finne avstanden mellom to noder i en vektet graf, og en sti mellom to noder med kortest mulig lengde.
- Dijkstras algoritme er også eksamensrelevant.

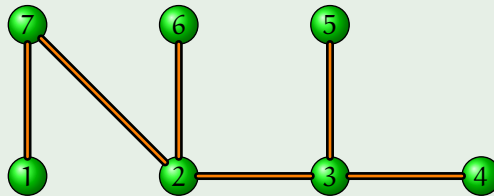
- Mot slutten av mandagens forelesning kom vi inn på trær med rot.
- Et tre med rot er et tre i grafteoretisk forstand hvor en av nodene er utpekt som rot.
- Et slikt tre tegner vi ofte ovenfra og ned, med roten øverst og de andre nodene lagvis nedover.
- Ofte kan vi sette merkelapper på nodene og på kantene.
- Vi så på noen eksempler på dette på mandag.

Trær med rot

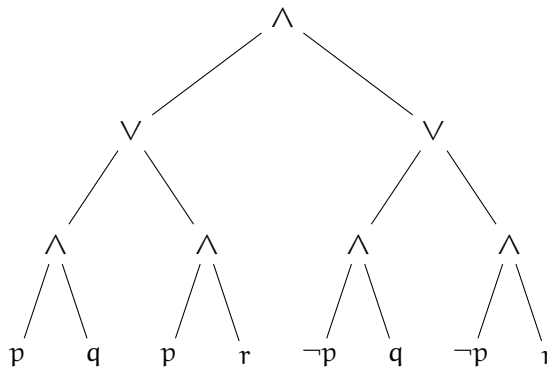
Hvis vi tar utgangspunkt i et tre uten rot, og så tegner det som et tre med rot, vil det visuelle resultatet avhenge mye av hvor vi plasserer roten.

Oppgave.

- Tegn følgende tre som et tre med rot når vi plasserer roten henholdsvis i nodene 1, 3 og 6:



- Mandag ga vi et eksempel på et syntakstre hvor vi løste opp et utsagnslogisk uttrykk i en trestruktur.



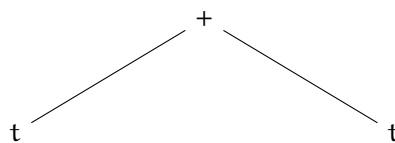
- Når vi skriver utsagnslogiske uttrykk eller algebraiske uttrykk, er vi vant med å skrive symbolene \wedge , \vee , $+$, \cdot og liknende mellom deluttrykkene.
- Denne skrivemåten heter med et fint ord for infiks, vi skriver symbolene innimellom teksten.
- Denne måten å skrive formler og algebraiske uttrykk på er historisk betinget, men den er nok også den måten som gir den best lesbare fremstillingen.
- Det finnes imidlertid andre måter som er vel så godt egnet for innmating i algoritmer, polsk; forlengs og baklengs.

- Fordelen med forlengs og baklengs polsk notasjon er at man slipper parenteser.
- Disse måtene å skrive uttrykk på er også bedre egnet når uttrykkene skal mates inn i en datamaskin.
- Programmeringsspråket *LISP* er basert på bruk av polsk notasjon, og i “gamle dager” måtte lommeregnerne programmeres med baklengs polsk notasjon.
- Brukere av editoren *emacs* vil oppdage at den innebygde kalkulatoren bruker baklengs polsk notasjon.
- La oss se på et eksempel.
- Vi skal gi en grammatikk som definerer alle termer i symbolene 0, 1, + og \times i forlengs polsk notasjon.
- En term er et uttrykk som beskriver et tall. Vi skal se på alle måter å beskrive tall på hvor vi bruker symbolene nevnt over.
- Poenget med polsk notasjon er at funksjonssymbol, logiske bindeord og andre symboler vi bruker til å binde sammen enkle uttrykk til mer komplekse settes først, og så kommer deluttrykkene etter, uten parenteser.
- En grammatikk er et sett regler som forteller oss hvilke ord som tilhører det formelle språket vi vil beskrive.
- I informatikk-litteratur har man utviklet en rask skrivemåte for slike grammatikker.

Term t

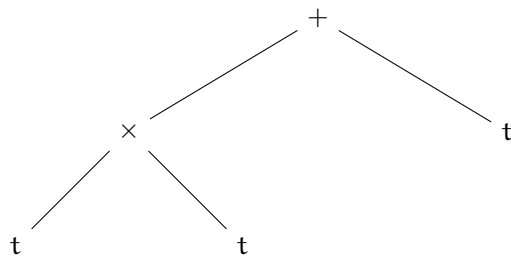
$t ::= 0 \mid 1 \mid ++t \mid \times t$

- Denne definisjonen skal leses som følger:
- Mengden av *termer* er den minste mengden som oppfyller
 - 0 og 1 er termer.
 - Hvis t og s er termer, er $++s$ og $\times s$ også termer.
- Vi har sett på tilsvarende konstruksjoner da vi så på formelle språk definert ved generell induksjon.
- En induktiv definisjon forteller oss at det ligger en trestruktur bak hvert ord i språket.
- Hvordan kan vi bruke trær til å bestemme om et ord i alfabetet $\{0, 1, +, \times\}$ er en term eller ikke, og hvordan kan vi bruke trær til å finne en mer lesbar form av termen?
- Er $\times 00 \times +100$ en term slik det ble definert på forrige side?
- Vi kan prøve å utvikle et syntakstre for dette ordet, hvor vi bruker bokstaven t for å markere at her må det stå en enklere term.
- Første tilnærming til syntakstreet må være

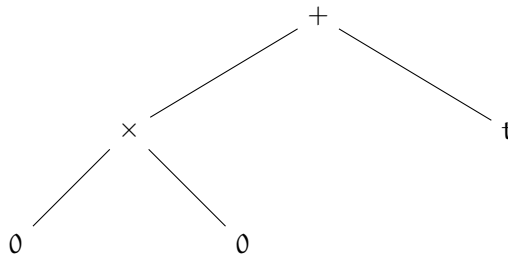


hvor $tt = \times 00 \times +100$

- Den første ukjente termen må begynne med \times , så syntakstreet må se ut som

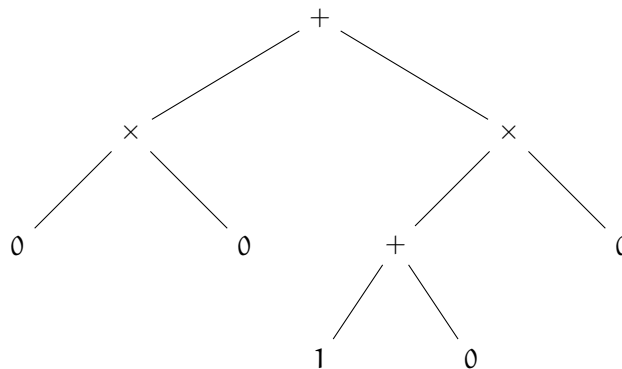


hvor $ttt = 00 \times + 100$. Her kan vi se direkte hva de tre t-ene må stå for, så vi får treet



hvor $t = \times + 100$.

- Vi kan fortsette å avsløre hvordan syntakstreet må se ut ved å lese problemordet vårt fra venstre mot høyre.
- Vi ser at neste term er et produkt hvor første faktor er summen av 1 og 0 og andre faktor er 0
(Vi er ikke interessert i verdien av denne termen, bare om det er en term).
- Det gir oss det fullstendige syntakstreet på neste side:



Skrevet på vanlig infiks-form får vi

$$(0 \times 0) + ((1 + 0) \times 0).$$

- Når man bruker baklengs polsk notasjon på en lommeregner, taster man inn tall og funksjonsuttrykk som $+$, \exp , \sin i rekkefølge.

- Hver gang man taster inn et funksjonsuttrykk, vil lommeregneren oppfatte siste tall, eller de to siste tallene, som input, og erstatte disse med funksjonsverdien.
- Det er altså funksjonsverdien som oppfattes som det siste tallet du tastet inn i fortsettelsen.
- Det er ikke vanskelig å se at lommeregneren kan behandle en sekvens av tall og funksjoner på bare en måte. Det betyr at et uttrykk i baklengs polsk notasjon bare kan tolkes på en måte.
- Det samme gjelder da selvfølgelig for forlengs polsk notasjon.
- *Syntakstreet* for en term eller et utsagnslogisk uttrykk er uavhengig av om vi har brukt vanlig infiks notasjon, forlengs polsk eller baklengs polsk notasjon.
- Det betyr at det må være mulig å finne disse uttrykkene på en systematisk måte fra syntakstreet.
- Dette er noe vi skal komme tilbake til når vi har snakket mer generelt om rekursive definisjoner over trær.
- Nå skal vi se hvordan vi kan formulere en algoritme som
 1. Avgjør om et ord w i alfabetet $\{+, \times, 0, 1\}$ er en term i forlengs polsk notasjon eller ikke.
 2. Finner syntakstreet til w når w er en term.
- Vi skal bruke en del uttrykk som blir gjort presise senere i forelesningene, men hvor meningen er så opplagt at det ikke bør lede til noen misforståelser.
- På denne måten forbereder vi oss også på innføringen av *binære trær* og *merkede trær*.
- Vi gir ikke en fullverdig pseudokode, men beskriver hvordan algoritmen skal utføres skritt for skritt.
- For enkelthets skyld antar vi at w ikke er det tomme ordet.
- På forelesningen ble algoritmen gjennomgått parallellt med et eksempel på tavla.

1 Lag en node. Denne første noden vil vi kalle *roten*.

2 Så lenge det finnes uleste bokstaver i w gjør vi følgende: [En **while**-løkke]

2.1 Hvis første uleste bokstav er 0 eller 1, merk noden med bokstaven.

Gå stien mot roten til du passerer en venstre-kant eller til du kommer til roten uten å ha passert noen venstre-kant.

Passerer du en venstrekant, lag et nytt barn til høyre for den noden du er på, flytt deg til denne barnenoden, begynn på neste bokstav i w og gå tilbake til skritt 2.

Treffer du noden uten å ha passert en venstrekant, aksepterer du w og har skrevet ferdig treet hvis det ikke er flere bokstaver igjen, mens du underkjenner w hvis det er flere bokstaver igjen.

2.2 Hvis første bokstav er + eller \times , merk noden med bokstaven.

Lag et nytt barn til venstre for den noden du er på, flytt deg til denne barnenoden, begynn på neste bokstav i w og gå tilbake til skritt 2.

3 Hvis det ikke finnes flere bokstaver i w og du ikke står på roten, underkjenner du w .

Vi skal teste algoritmen på ordene

$+ \times +010$

og

$+0 \times +010$

på tavla.

- Vi har nå sett på noen eksempler på trær med rot.
- Det er på tide å gjøre noe av den terminologien vi bruker når vi snakker om trær med rot mer presis.

Definisjon.

La T være et tre med rot.

- Med nivået til en node mener vi antall kanter mellom noden og roten.
- Hvis det finnes en kant mellom node a og node b og a har lavest nivå (ligger øverst i tegningen) sier vi at b er barnet til a .
- Hvis det finnes en sti mellom to noder a og b som ikke går via roten, vil den som har det høyeste nivået (ligger lavest i tegningen) være etterkommer til den andre, som omvendt er forgjenger til den første.

Definisjon (Fortsatt).

- En node som ikke har noen barn er et blad.
- En gren er en sti fra roten til et blad.
Noen vil kalle dette en maksimal gren og la en gren være en sti fra roten til en node.

Oppgave.

Vis at det finnes en bijeksjon mellom mengden av blader og mengden av grener i et tre med rot.

Binære trær

- De aller fleste trærne vi har sett på, og til nå alle de som har kommet ut av informative eksempler, har den egenskapen at hvis en node ikke er en bladnode, har den nøyaktig to barn.
- Det finnes mange trær som ikke har den egenskapen, eksempelvis familietreet til Harald Hårfagre, men det er så mange som har egenskapen at vi skiller dem ut ved en egen betegnelse, og ved å legge ekstra struktur på dem:

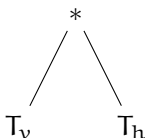
Definisjon.

Et binært tre er et tre med rot slik at

- Enhver node er enten en bladnode eller har nøyaktig to barn.
- Hvis en node har to barn, vil det ene barnet betegnes som barnet til venstre og det andre som barnet til høyre

- Hvis T er et binært tre, kan vi snakke om venstre deltre og høyre deltre.
- Det venstre deltreet får vi ved å fjerne roten, barnet til høyre og alle etterkommerne av det.
- Tilsvarende får vi det høyre deltreet ved å fjerne roten, barnet til venstre og alle etterkommerne til det.
- I venstre (høyre) deltre blir da barnet til venstre (høyre) den nye roten.
- I treet hvor roten også er et blad, kan vi ikke snakke om venstre eller høyre deltre.

- Snur vi på dette, kan vi oppfatte mengden av binære trær som induktivt definert.
- Utgangspunktet, eller *induksjonstarten* er nulltreet, treet som består av en node og ingen kanter.
- Denne noden er da både *rot* og *blad*.
- *Induksjonskrittet* består i at vi tar to binære trær T_v og T_h , en ny rotnode og to nye kanter, mot venstre til roten i T_v og til høyre mot roten i T_h .



Sammensetning av to binære trær til ett.

Noen konkrete eksempler vises på tavla.

- Vi kan oppfatte denne sammensetningen som en form for *sum* av to binære trær, vi legger sammen to trær til et nytt tre.
- Vi kan godt skrive $T_v \oplus T_h$ for denne sammensetningen av trær.
- Merk at den kommutative loven ikke gjelder, det betyr mye hvilket av trærne som settes til venstre og hvilket som settes til høyre.
- Som grafer betyr det ikke så mye, men for trerekursjon betyr det endel, siden vi der kan referere til venstre og høyre deltre.
- Vi skal nå se på en form for produkt av trær.
- Vi skriver $*$ for null-treet.

Definisjon.

Vi definerer "treproduktet" \otimes ved

- $* \otimes S = S$
- $(T_v \oplus T_h) \otimes S = (T_v \otimes S) \oplus (T_h \otimes S)$

Poenget med å gi denne definisjonen er å gi et eksempel på hvordan man kan definere ting ved rekursjon på oppbyggingen av et tre.

Vi illustrerer hva som skjer ved et par eksempler på tavla.

Vi så at effekten er å erstatte alle bladnodene i T med kopier av S .

- Det er en intim sammenheng mellom bitsekvenser og binære trær.
- For hver node i i et binært tre kan vi lage oss en tilsvarende bitsekvens ved rekursjon på nivået (avstanden til roten) til noden:
- $\text{bit}(*)$ er den tomme sekvensen hvis $*$ er rotnoden.
- Hvis a er en node med to barn, b til venstre og c til høyre, og $\text{bit}(a) = x_1 \cdots x_k$, lar vi
 - $\text{bit}(b) = x_1 \cdots x_k 0$.
 - $\text{bit}(c) = x_1 \cdots x_k 1$.

Denne definisjonen illustreres på tavla.

- Omvendt vil enhver endelig mengde X av bitsekvenser, eller 0-1-sekvenser, bestemme et binært tre hvor vi først ser på alle delsekvenser av sekvensene i X , så lar bladnodene være de minimale bitsekvensene som ikke er delsekvens av noen sekvens i X og til slutt organiserer dette til et tre ved å la den tomme sekvensen bli roten, og så gå til venstre eller høyre avhengig av om neste bit er 0 eller 1.
- Vi illustrerer denne konstruksjonen på tavla.

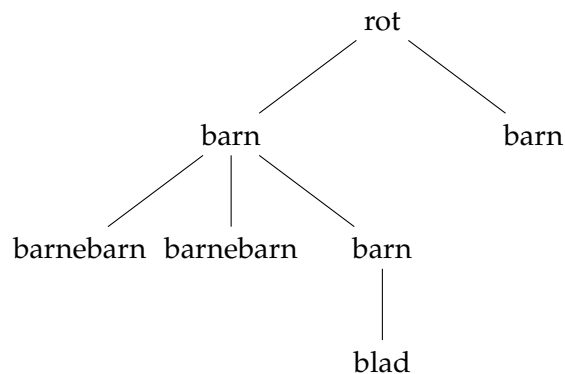
Forelesning 26

Trær

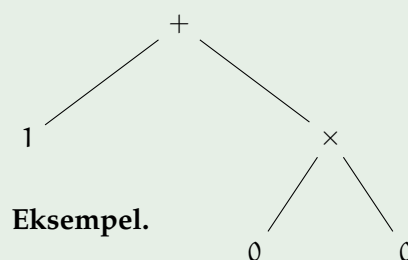
Dag Normann - 28. april 2008

Oppsummering

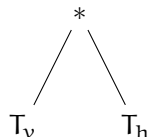
- Sist forelesning snakket vi i hovedsak om trær med rot, og om praktisk bruk av slike.



- Her er noen av de begrepene vi innførte.
- Som en viktig klasse trær så vi på *syntakstrær*, det vil si trær som fanger opp oppbyggingen av en formel eller en term.
- Poenget er at det er den logiske oppbyggingen som fanges opp, hvordan et uttrykk er sammensatt av enklere deluttrykk.
- Vi så på tre tradisjonelle måter å skrive uttrykk på papir eller som symbolsekvenser på:
 - *Infiks*, den vanlige måten hvor symbolet som kombinerer to uttrykk skrives imellom.
Dette er den vanlige måten vi har brukt hele livet, og som krever bruk av parenteser.
 - *Forlengs polsk* hvor vi eksempelvis skriver $+ts$ i stedet for $t + s$.
 - *Baklengs polsk* hvor vi eksempelvis skriver $ts+$ i stedet for $t + s$.

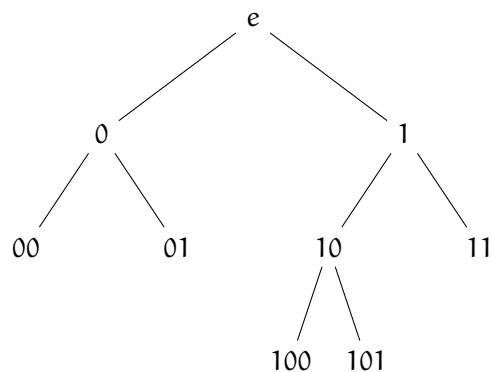


- Vanlig: $1 + (0 \times 0)$
 - Polsk: $+1 \times 00$
 - Baklengs polsk $100 \times +$
-
- Vi så på en algoritme for hvordan man kan bygge opp et syntakstre fra et uttrykk hvor vi har brukt polsk notasjon.
 - Denne algoritmen avgjør også om et ord faktisk er en term hvor vi har brukt polsk notasjon eller om den ikke er det.
 - Det er ikke meningen at dere skal kunne følge denne algoritmen skritt for skritt, men det er meningen at dere skal kunne bestemme om et uttrykk er en term skrevet på polsk, og at dere skal kunne finne syntakstreet og kunne skrive den eventuelle termen med infiks notasjon.
 - Vi vil regne noen eksempeloppgaver rundt dette før vi sier oss ferdige med avsnittet om trær.
-
- Vi så også på binære trær.
 - Et *binært tre* er et tre hvor hver node enten er en bladnode eller har nøyaktig to barn.
 - I et binært tre vil vi skille mellom venstre barn og høyre barn når en node har to barn.
 - Det betyr at matematisk sett har vi lagt mere struktur på et binært tre enn bare det å utpeke en node som rot.
-
- Vi så på de binære trærne som en *induktivt definert* mengde av matematiske objekter.
 - *Null-treet*, det som bare består av en node, og denne er både rot og blad, er et binært tre.
 - Hvis T_v og T_h er to binære trær, kan vi sette dem sammen til et nytt binært tre $T =$



- Vi kan da definere en funksjon f ved rekursjon over oppbyggingen av binære trær ved
 - Bestemme hva $f(*)$ er.
 - Bestemme hvordan $f(T)$ avhenger av de to deltrærne $f(T_v)$ og $f(T_h)$ når T er et sammensatt tre.
 - Vi så hvordan vi kan definere produktet av to trær ved rekursjon.
 - I dag skal vi se på flere anvendelser av trerekursjon.
 - Trerekursjon kan være en oversiktlig måte å beskrive en funksjon på, men skriver man et program basert på trerekursjon, kan det arbeide ubehagelig langsomt.
-
- Vi så på hvordan vi kan finne en bit-sekvens til hver node i et binært tre, og vi så på hvordan vi kan lage et binært tre fra en mengde bit-sekvenser.
 - Dette siste er ikke så viktig at vi bruker mer tid på det, men vi skal se på et eksempel som viser hvordan vi markerer nodene i et binært tre med bitsekvenser.

Binære trær



Binære trær

- En digital strøm er en uendelig følge $\{x_n\}_{n \in \mathbb{N}}$ hvor hver x_i er en bit, markert som 0 eller 1.
- En digital strøm kan oppfattes som en strøm av data på digital form.
- La oss anta at vi har en prosedyre hvor input kan være en digital strøm og hvor output er en eller annen melding på digital form.
- Det vil finnes situasjoner hvor vi aldri får noe output hvis input er spesielt ekle digitale strømmer, men normalt vil vi at prosedyren skal avsløre om den digitale strømmen vi mottar er uten interesse, og skal avslutte med en melding om det.

- Vi tenker oss altså en situasjon hvor prosedyren avslutter med et svar uansett hvilken datastrøm den fores med.
- For enhver datastrøm finnes det da en endelig del som er stor nok til at prosedyren vår kan gi et output på grunnlag av denne.
- Det er fordi prosedyren vår bare kan utnytte endelig mye informasjon om ver enkelt datastrøm.
- La T være treet av endelige bitsekvenser som er så små at prosedyren vår ikke har nok grunnlag i disse til å gi et output.
- Er T et endelig tre?

- Vi skal vise at det er tilfelle.
- Beviset er et eksempel på et kontrapositivt bevis, altså på et bevis hvor vi antar at konklusjonen er feil, og resonnerer oss frem til at da er premissene feil.
- Anta derfor at treet er uendelig.
- Da må venstre deltre være uendelig eller høyre deltre være uendelig.
- Start en digital strøm med 0 om venstre deltre er uendelig, og med 1 om venstre deltre er endelig. La T_1 være det tilsvarende uendelige deltreet.
- Vi velger neste bit i datastrømmen som 0 om venstre deltre i T_1 er uendelig, og som 1 om venstre deltre i T_1 er endelig. Da er høyre deltre i T_1 uendelig.

- Slik fortsetter vi ved å gå til venstre når deltreet i den retningen er uendelig, og til høyre når det er nødvendig for fortsatt å ha et uendelig deltre.
 - På den måten bygger vi opp en digital strøm som prosedyren vår ikke kan gi noe output fra, for da ville den gjøre det fra en endelig del av strømmen.
 - Vi har imidlertid sørget for at enhver endelig del av den strømmen vi konstruerer ligger i T , og derfor er utilstrekkelig for dette.
-
- Påstanden vi viste på forrige side har den praktiske konsekvensen at hvis vi først har greid å lage en prosedyre som gir et svar uansett hvilken digital strøm vi forer den med, så finnes det en øvre grense for hvor lenge vi må vente på et svar, uavhengig av hva input er.
 - Dette er et eksempel på en påstand hvor vi må gi et indirekte bevis, eller i det minste gå utenom den konstruktive delen av matematikken.
 - Dette er ikke noe tema i MAT1030, og vi skal ikke forfølge dette aspektet videre.
-
- Hvordan skal vi så kunne avgjøre om et tre uten rot kan være et binært tre stripet for all ekstra struktur?
 - Nulltreet med bare en node er et binært tre.
 - For andre binære trær vil
 1. Roten ha grad to.
 2. Bladene ha grad 1
 3. Alle andre noder ha grad 3.
-
- Omvent, hvis T er et tre uten rot, slik at
 1. En node har grad 2
 2. Alle andre noder har grad 1 eller grad 3
 så kan vi organisere T til et binært tre.
 - Nivået til en node blir da avstanden til noden av grad 2, som blir roten.
 - Vi står fritt i å velge hva som skal ligge til høyre og hva som skal ligge til venstre.
 - Overbevisende eksempler vises på tavla.

Merkede trær

- Vi skal nå gå tilbake til syntakstrær, og se på hvordan vi kan lese *infiks*-notasjonen og de to polske notasjonene ut av et slikt tre.
- Alle tre prosessene kan beskrives ved hjelp av en rekursiv prosedyre, hvor vi bruker trerekursjon.
- Som vi husker, markerte vi nodene i syntakstreet med symboler, vi skrev 0 eller 1 på bladene, og vi skrev + eller \times på foreldrenodene.
- På engelsk brukes ordet labels om slike merkelapper på nodene.
- Vi skal la et merket tre være et tre hvor vi har markert hver node med et symbol eller en tekst.

- Syntakstreet for formelen

$$A = ((p \wedge q) \vee (p \wedge r)) \wedge ((\neg p \wedge q) \vee (\neg p \wedge r))$$

var et slikt merket tre, og syntakstreet for ordet

$$+ \times 00 \times +100$$

var et annet eksempel.

- Når vi skal studere bruken av merkede trær, vil vi ofte begrense hvilke merkelapper vi kan bruke på bladnodene og hvilke merkelapper vi kan bruke på foreldrenodene.

Eksempel.

- Hvis vi merker foreldrenodene med \vee eller \wedge og bladnodene med p , $\neg p$, q , $\neg q$, r eller $\neg r$, vil treet representere et utsagnslogisk uttrykk på svak normalform.
- Hvis vi merker foreldrenodene med $+$ eller \times og bladnodene med 0 , 1 eller -1 får vi termer som kan uttrykke elementer i \mathbb{J} på forskjellig vis.

- Vi skal ta for oss det siste eksemplet, det fra heltallsteori, og se hvordan vi kan definere henholdsvis
 - funksjonen infiks som gir oss den vanlige måten å skrive en term på,
 - funksjonen polsk som gir oss termen med polsk notasjon,
 - funksjonen revpolsk som gir oss termen på baklengs polsk form,
- og vi skal vise ved eksempler på tavlen hvordan disse rekursjonene virker.

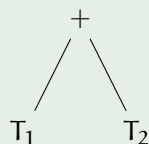
Eksempel.

Vi definerer funksjonen $\text{infiks}(T)$ ved trerekursjon ved

- Hvis roten i T er en bladnode med merke a ($a = 0$, $a = 1$ eller $a = -1$) lar vi

$$\text{infiks}(T) = a$$

- Hvis T er på formen



lar vi

$$\text{infiks}(T) = (\text{infiks}(T_1) + \text{infiks}(T_2)).$$

- Tilsvarende for \times .

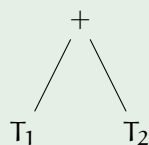
Eksempel.

Vi definerer funksjonen $\text{polsk}(T)$ ved trerekursjon ved

- Hvis roten i T er en bladnode med merke a ($a = 0$, $a = 1$ eller $a = -1$) lar vi

$$\text{polsk}(T) = a$$

- Hvis T er på formen



lar vi

$$\text{polsk}(T) = +\text{polsk}(T_1)\text{polsk}(T_2).$$

- Tilsvarende for \times

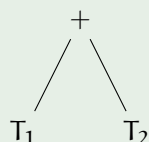
Eksempel.

Vi definerer funksjonen $\text{revpolsk}(T)$ ved trerekursjon ved

- Hvis roten i T er en bladnode med merke a ($a = 0$, $a = 1$ eller $a = -1$) lar vi

$$\text{revpolsk}(T) = a$$

- Hvis T er på formen



lar vi

$$\text{revpolsk}(T) = \text{revpolsk}(T_1)\text{revpolsk}(T_2) + .$$

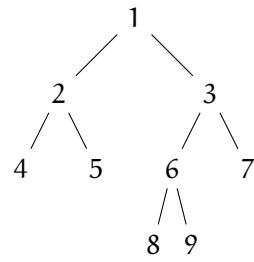
- Tilsvarende for \times .

- De tre rekursive definisjonene svarer til eksemplene i læreboka på *in-order* (infiks), *pre-order* (polsk) og *post-order* (revpolsk) traversering av treet.
- En traversering av treet innebærer at vi leser nodene i treet i en bestemt rekkefølge, og utfører operasjoner (som å skrive symboler) i en bestemt rekkefølge.
- Alle disse funksjonene vil benytte en ovenfra og ned traversering i den forstand at vi leser treet fra toppen og nedover. Vi skal ikke si så mye om traversering her.
- Det er imidlertid slik at en rekursiv konstruksjon gir en mer eller mindre forstandig måte å frembringe et resultat på, i den forstand at vi må lagre mindre eller mer informasjon underveis for å oppnå resultatet.

- Det skal vi se nærmere på for de tre funksjonene.
- Hvis vi skal beregne $\text{polsk}(T)$ har vi en enkel oppgave.
- Vi starter med å lese merket på rotnoden, skriver den (uten å måtte huske hva det var) og beregner så $\text{polsk}(T_v)$ og $\text{polsk}(T_h)$.
- Denne algoritmen vil lese nodene i T i en bestemt rekkefølge og skrive ut merkene i den samme rekkefølgen.
- Vi trenger altså ikke å sette av noe plass til hukommelse for å beregne denne funksjonen, og det finnes ingen mer effektiv rekkefølge å lese treet på enn den som følger fra den rekursive konstruksjonen.
- Funksjonen revpolsk er basert på at vi leser nodene i syntakstreet i den samme rekkefølgen, ovenfra og ned, og barnenodene før søskennodene, men ellers, fra venstre mot høyre.
- Siden merket på roten skal skrives til sist, må dette merket lagres, normalt i noe vi kaller en stack, og dette symbolet skriver vi bare ut når vi er helt ferdige med resten.
- Da må vi selvfølgelig lagre informasjon under beregningen av $\text{revpolsk}(T_v)$ og av $\text{revpolsk}(T_h)$ også.
- Hvis vi starter med å lese bladet nederst til venstre, og så leser treet fra venstre mot høyre, dog slik at vi leser foreldrene når søskenflokken er ferdiglest, vil vi lese nodene treet i den rekkefølgen vi skriver dem.
- En slik rekkefølge kalles en nedenfra og opp-gjennomgang av treet (*traversering* er et finere ord for gjennomgang).
- Beregningen av infiks er langt på vei den tyngste.
- Som før leser vi treet ovenfra og ned, og fra venstre mot høyre.
- Vi må lagre merkene på nodene underveis, men bare mens vi behandler det venstre deltreet.
- Det som kompliserer algoritmen er at når vi skal skrive ned merket på en foreldrenode, må vi også skrive parenteser på de stedene de skal stå.
- Dette innebærer at vi ikke kan konstruere den digitale formen av sluttproduktet som en enkel rekursiv prosess hvor nye bits føyes til i enden, men at vi må putte nye bits innimellom de vi allerede har skrevet.
- Det er ingen naturlig måte å lese nodene i T på slik at vi kan skrive $\text{infiks}(T)$ ned fra venstre mot høyre etterhvert som vi leser.
- Mennesker liker infiksnotasjonen, men maskiner gjør det ikke.
- Det kan skyldes at en menneskehjerne er en parallellprossessor, det vil si at den håndterer flere informasjonsbiter på en gang, mens dagens datamaskiner fortsatt arbeider sekvensielt.
- Ettersom alt dette er spekulasjoner, og ikke matematikk, går vi over til et annet tema.

Traverseringer

- Vi skal ikke si så mye mer om traverseringsrekkefølger av trær, men se på et generelt eksempel:



- 1,2,3,4,5,6,7,8,9: Bredde først.
- 1,2,4,5,3,6,8,9,7: Skrive polsk.
- 4,5,2,8,9,6,7,3,1: Skrive baklengs polsk.
- 4,2,5,1,8,6,9,3,7: Infiks-rekkefølgen
- Mange andre muligheter.

Forelesning 27

Trær

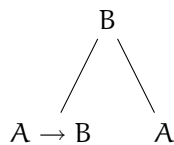
Dag Normann - 30. april 2008

Oppsummering

- Mandag så vi på hvordan vi kan finne uttrykk og termer på infiks form, med polsk notasjon og med omvendt polsk notasjon fra syntakstreet.
- Vi skal repetere de viktigste aspektene ved dette gjennom endel eksempeloppgaver.
- I dag skal vi se på det som kalles *bevistrær*, men mest på *unifisering* og *unifiseringsalgoritmer*

Merkede trær

- En annen type trær som spiller en stor rolle i logikk, men også i informatikk, er bevistrærne.
- Et bevistre er et merket tre hvor hver node er merket med en formel.
- Bladnodene vil være opplagt sanne, aksiomer i en eller annen formalisert teori, og merket til en foreldrenode vil følge logisk fra merkene til barna ut fra visse prinsipper.
- En mulighet er at vi kan få lov til å ha deler av et bevistre som ser ut som



som uttrykker at hvis vi har bevist A og $A \rightarrow B$, så kan vi konkludere B .

- Et slikt bevistre vil da være en garantist for at formelen som merker roten må være en konsekvens av de aksiomene som er brukt.
- Vi skal ikke la dette utvikle seg til et kurs i logikk, eller bevisteori, men som et eksempel på bruk av trær, skal vi se hvordan vi kan finne et bevis for et utsagnslogisk uttrykk, et tre som vil være en garantist for at uttrykket er en tautologi.
- Det er et poeng at hvis uttrykket er på *svak normalform*, så har vi en prosedyre for å omforme syntakstreet til et forsøksvis bevistre, og alle bladene blir aksiomer nøyaktig når utgangspunktet var en tautologi.
- Vi minner om at et utsagnslogisk uttrykk er på svak normalform hvis vi
 - Bare bruker bindeordene \wedge , \vee og \neg .
 - \neg kan bare stå rett foran en utsagnsvariabel.
- Som eksempler på rekursive konstruksjoner som går ut over rekursjon over \mathbb{N} så vi på hvordan vi systematisk kan

- fjern forekomster av \rightarrow og \leftrightarrow i et utsagn slik at vi får et ekvivalent utsagn med bare \wedge , \vee og \neg .
- som simultanrekursjon, skriv om utsagnene A og $\neg A$ til svak normalform.
- Disse konstruksjonene kan også formuleres ved hjelp av rekursjon på syntakstrær.
- Siden vi begrenser oss til utsagn på svak normalform, kan vi ikke bruke den regelen vi ga eksempel på, ettersom \rightarrow ikke inngår i vokabularet.
- Vi skal tillate en type forgrening, eller slutningsregel:

$$\begin{array}{ccc}
 & A_1 \vee \dots \vee A_n \vee (B \wedge C) & \\
 & \swarrow \quad \searrow & \\
 A_1 \vee \dots \vee A_n \vee B & & A_1 \vee \dots \vee A_n \vee C
 \end{array}$$

- Vi skal ikke la rekkefølgen av delutsagnene i et \vee -utsagn bety noe.
- De eneste bladnodene vi vil akseptere er noder med merkene

$$A_1 \vee \dots \vee A_n \vee p \vee \neg p$$

hvor p er en utsagnsvariabel,

altså disjunksjoner som inneholder både en utsagnsvariabel og negasjonen dens.

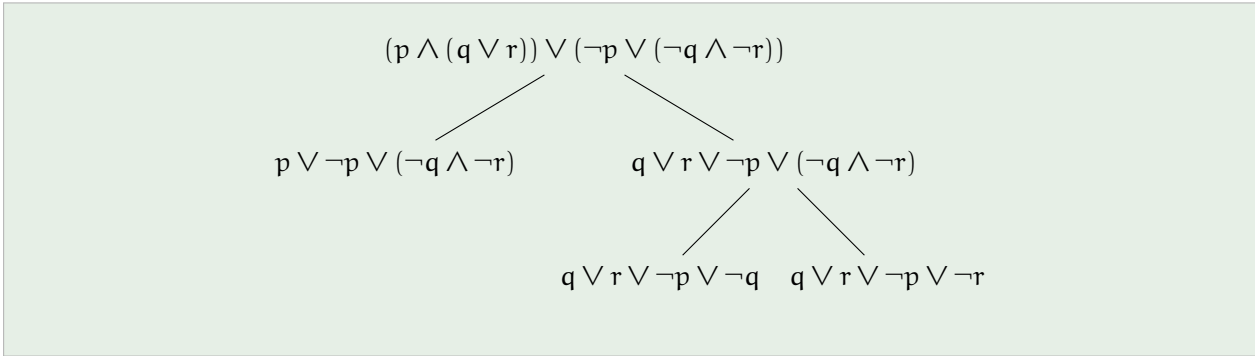
- Slike disjunksjoner er opplagt tautologier, og vil tjene som aksiomer.
- Vi skal se på et par eksempler hvor vi starter med et utsagn og utvikler et bevis for dette utsagnet.
- Eksemplene forklares på tavla, om nødvendig.

Eksempel.

$$\begin{array}{ccc}
 & (p \wedge q) \vee (\neg p \vee \neg q) & \\
 & \swarrow \quad \searrow & \\
 p \vee \neg p \vee \neg q & & q \vee \neg p \vee \neg q
 \end{array}$$

- Når vi bruker den distributive loven på $p \wedge q$ får vi en konjunksjon av to opplagte tautologier.
- Tautologier på denne formen har vi kalt aksiomer.
- Resultatet blir et bevis.

Eksempel.



La oss nå se på prosedyren for å lage et forsøksvis bevistre fra et utsagnslogisk uttrykk. Hvis vi har et utsagn A på svak normalform, kan A skrives på formen

$$A = A_1 \vee \dots \vee A_n$$

hvor $n \geq 1$ og hver A_i enten er en konjunksjon, en utsagnsvariabel p eller negasjonen $\neg p$ av en utsagnsvariabel.

Hvis det finnes i og j slik at A_i er en utsagnsvariabel, og A_j er negasjonen av den samme, lar vi roten være bladnode merket med A , og vi har et bevistre.

Hvis alle A_i er utsagnsvariable eller negasjonen av slike, men vi er ikke i situasjonen over, lar vi roten være en bladnode, men konkluderer med at vi ikke har noe bevistre.

Ellers går vi til den minste i slik at $A_i = C \wedge D$.

Da lager vi to barn, hvor vi erstatter A_i med C når vi bygger treet videre under det ene barnet og vi erstatter A_i med D når vi bygger treet videre under det andre barnet.

Hvis begge disse deltrærne blir bevistrær, har vi konstruert et bevistre, ellers har vi ikke gjort det.

Vi illustrerer prosedyren på utsagnet

$$(\neg p \vee (p \wedge (p \vee q))) \wedge (p \vee \neg q \vee (q \wedge \neg p))$$

på tavla.

- Vi skal avslutte dette avsnittet om hvor trær brukes med å se på en teknikk som kalles unifisering.
- Hvis vi har to termer hvor det forekommer variable, er det da mulig å erstatte disse variablene med andre termer slik at resultatene blir like?
- Vi skal begrense oss til et enkelt tilfelle, men den generelle unifiseringsalgoritmen spiller en stor rolle i logikkprogrammering og i automatisk bevissøk.

Unifisering

Eksempel.

- Kan vi unifisere termene

$$(x \times (1 + 0)) + (((0 + 1) + z) \times (1 + x))$$

og

$$((0 + 1) \times (z + 0)) + ((y + 1) \times (1 + x))$$

det vil si, kan vi finne andre termer vi kan sette inn for x , y og z slik at de to uttrykkene blir like?

- Dette kan vi gjøre ved å sammenlikne termene systematisk og se om vi får fremtvunget hva vi skal erstatte x , y og z med for at resultatet skal bli vellykket.

Eksempel (Fortsatt).

- Først ser vi at begge termene har $+$ som hovedsymbol (dette ser vi lettere ut fra syntakstreet), og skal vi unifisere termene, må vi unifisere termene

$$(x \times (1 + 0)) \text{ og } ((0 + 1) \times (z + 0))$$

$$(((0 + 1) + z) \times (1 + x)) \text{ og } ((y + 1) \times (1 + x))$$

simultant, det vil si vi må sette inn de samme termene for x , y og z i begge tilfellene.

Eksempel (Fortsatt).

Vi ser at begge termene i det første paret er produkttermer og det samme gjelder for begge termene i det andre paret. Oppgaven blir derfor å unifisere alle disse fire parene simultant:

1. x og $0 + 1$
2. $1 + 0$ og $z + 0$
3. $(0 + 1) + z$ og $y + 1$
4. $1 + x$ og $1 + x$

Vi ser at linje 1 forteller oss at vi må sette inn $0 + 1$ for x og i linje 4 har vi to like termer. De to andre linjene løser seg opp i fire nye linjer:

Eksempel (Fortsatt).

Vi må kunne unifisere

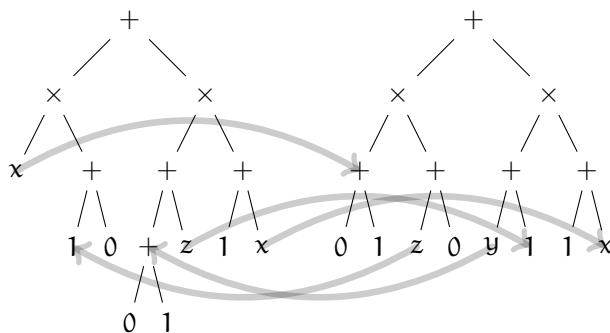
1. 1 og z
2. 0 og 0
3. 0 + 1 og y
4. z og 1

simultant.

Nå har vi nådd bunnen av det som blir rekursjonen, og vi ser at hvis vi setter inn 1 for z og 0 + 1 for y, i tillegg til at vi skulle sette inn 0 + 1 for x, så får vi unifikasjon av de to første termene.

$$(x \times (1 + 0)) + (((0 + 1) + z) \times (1 + x))$$

$$((0 + 1) \times (z + 0)) + ((y + 1) \times (1 + x))$$



- $x = (0 + 1)$
- $z = 1$
- $y = (0 + 1)$
- Vi skal se på flere illustrerende eksempler før vi beskriver den endelige algoritmen for unifikasjon.
- Unifikasjonen vil være mislykket hvis vi på et trinn må unifikere to forskjellige termer uten fri variable.
- Det er hele tiden viktig å tenke på at det er de syntaktiske uttrykkene som skal bli like, ikke bare de numeriske verdiene.
- For at en maskin skal kunne teste om en overgang er logisk korrekt, må uttrykk som spiller rollen som utsagnsvariable være syntaktisk like.
- Unifikasjon spiller derfor en viktig rolle i logikkprogrammering, det vil si, når man arbeider med programmeringsspråk som PROLOG og ML basert på logikk.

Eksempel.

- Unifiser $(1 + x) + (y + z)$ og $(1 + y) + (x + 0)$.
- Dette krever, i to trinn, at vi skal kunne unifisere
 1. 1 og 1
 2. x og y
 3. y og x
 4. z og 0simultant.
- Bare vi setter inn den samme termen for x , y og vi setter inn 0 for z , får vi en unifisering uansett.
- Vi vil derfor falle ned på den mest generelle unifiseringen, hvor vi beholder en av variablene x og y .
- Svaret blir derfor $(1 + x) + (x + 0)$

Eksempel.

- Vi skal unifisere $(1 + x) + (1 + y)$ og $(1 + 0) + (1 + (x + 1))$
- Dette reduseres til oppgaven å skulle unifisere
 1. $1 + x$ og $1 + 0$
 2. $1 + y$ og $1 + (x + 1)$simultant.

Eksempel (Fortsatt).

- Disse løser seg opp i fire oppgaver om å unifisere
 1. 1 og 1
 2. x og 0
 3. 1 og 1
 4. y og $x + 1$simultant.
- Linjene 1 og 3 er uproblematisk.
- I linje 2 ser vi at vi må sette inn 0 for x
- Det betyr at vi må omforme linje 4 til å skulle unifisere y og $0 + 1$, noe vi kan gjøre på direkten.

Eksempel.

- Anta at vi skal prøve å unifisere $x + (y + 0)$ og $(x + 1) + (1 + 0)$.
- Da må vi kunne unifisere
 1. x og $x + 1$
 2. $(y + 0)$ og $(1 + 0)$simultant.
- Andre linje er uproblematisk, men første linje er umulig, det finnes ingen term t slik at t og $t + 1$ er syntaktisk like.
- Hvis vi i forsøket på å unifisere par av termer må erstatte en variabel med en større term hvor variabelen forekommer, må vi konkludere med at unifisering er umulig.
- Ser vi på eksemplet $x + x$ og $y + (y + 1)$ reduseres det til samme type umulighet.

- Vi skal nå beskrive en rekursiv prosess som avgjør om det er mulig simultant å unifisere en endelig mengde par av termer i språket vårt.
- Etter at vi har gjort det, skal vi se på et eksempel på hvor vi kan få bruk for unifisering, og hvordan vi må tilpasse en konkret situasjon til en som kan håndteres av algoritmen vår.
- Vi vil starte med en endelig liste x_1, \dots, x_k av variable og to lister t_1, \dots, t_n og s_1, \dots, s_n av termer hvor disse variablene kan forekomme, og vi vil bestemme om det er mulig å erstatte variablene med termer r_1, \dots, r_k slik at hver t_i blir lik sin makker s_i .
- Får vi til det, sier vi at vi unifiserer parene simultant.

- Hvis t_1 er en variabel x_i og x_i forekommer i s_1 , men s_1 er mer kompleks, gi opp unifiseringen. Det samme gjelder om situasjonen er omvendt.
- Hvis t_1 er variabelen x_i og x_i ikke forekommer i s_1 , noterer vi at vi skal bruke sluttverdien av s_1 som termen r_i som skal erstatte x_i .

Deretter erstatter vi alle andre forekomster av x_i med s_1 og fortsetter unifiseringsalgoritmen.

I dette tilfellet har vi oppnådd å redusere antall variable med 1.

Et korrekthetsbevis for algoritmen vil i første omgang være ved induksjon over antall variable.

- Hvis t_1 er en variabel, mens s_1 ikke er det, fortsetter vi på tilsvarende måte.
- Hvis s_1 og t_1 er samme term, bare stryker vi dette paret fra listen og fortsetter.
- Hvis ingen av tilfellene over gjelder har vi to muligheter:
 - s_1 og t_1 er åpenbart forskjellige, eksempelvis ved at den ene er en sum og den andre et produkt, den ene er et tall mens den andre er et funksjonsuttrykk eller de er forskjellige tall.
I dette tilfellet konkluderer vi med at unifisering er umulig.
 - De er begge en sum eller de er begge et produkt.

Da erstatter vi paret s_1, t_1 med to par av mindre uttrykk, paret av de første addendene (faktorene) og paret av de andre addendene (faktorene) slik vi har sett eksempler på.

Deretter fortsetter vi algoritmen fra start.

- Hvis vi ikke finner ut underveis at unifikasjon er umulig, vil denne fremgangsmåten finne frem til de mest generelle termene r_1, \dots, r_k vi kan erstatte x_1, \dots, x_n med for å unifikere alle parene simultant.
- Korrekthetsbeviset er ved induksjon over antallet k av variable, med en underinduksjon over antall symboler sammenlagt i de to listene (dette antallet kan øke når vi kvitter oss med en variabel).
- Det finnes andre, og i praksis mer effektive, måter å gjennomføre unifikasjon på, hovedpoenget her var å vise prinsippene bak algoritmen.

Det er ikke meningen at dere skal kunne gjengi denne algoritmen, men at dere skal kunne bestemme for hånd, i forholdsvis enkle tilfeller, om termer lar seg unifikere, og i tilfelle, gjennomføre det.

Det kan i det minste være en fordel til eksamen å vite hva unifikasjon innebærer.

Nå skal vi svare på spørsmålet om hva dette skal være godt for.

Som tidligere nevnt spiller unifikasjon en rolle i automatisk bevisøk, eksempelvis i forbindelse med PROLOG.

Vi skal se på et eksempel på hvordan vi systematisk kan søke etter et bevis for en påstand i et veldig enkelt logisk system.

Eksemplet er så enkelt at vi ikke trenger hjelp av datamaskin til å finne et bevis, så det er mest til informasjon og motivasjon.

- Anta at vi har to aksiomer som vi kan bruke til å bevise at enkelte termer beskriver mindre tall enn andre termer:
 - $x < x + 1$
 - $x < y \wedge y < z \rightarrow x < z$.
- Så ønsker vi å søke etter et bevis for at

$$1 < ((1 + 1) + 1) + 1.$$

- Vi kan ikke finne en unifikasjon med aksiom 1, så håpet må være at vi har kommet frem til denne ulikheten som en anvendelse av aksiom 2.
- PROLOG vil da unifikere problemet vårt med konklusjonen i aksiom 2, og det er trivielt $x = 1$ og $z = ((1 + 1) + 1) + 1$.
- Dette følger fra $1 < y \wedge y < ((1 + 1) + 1) + 1$, og PROLOG vil lete etter en verdi for y slik at begge delene av denne konjunksjonen kan bevises.
- $1 < y$ lar seg unifikere med aksiomet $x < x + 1$ ved å la $x = 1$ og $y = 1 + 1$.
- Prøver vi denne veien, må vi også prøve å bevise den andre delen av konjunksjonen for denne verdien av y , nemlig at $1 + 1 < ((1 + 1) + 1) + 1$.
- Igjen ser vi at dette ikke kommer direkte fra aksiom 1, så skal vi kunne bevise denne påstanden, må det være som en konsekvens av aksiom 2 og et bevis for en instans av

$$1 + 1 < y \wedge y < ((1 + 1) + 1) + 1.$$

- Setter vi inn $(1 + 1) + 1$ for y , i et forsøk på å la første del av denne konjunksjonen være en direkte konsekvens av aksiom 1, ser vi at også andre del blir en direkte konsekvens av aksiom 1.
- Vi har dermed systematisk lett oss frem til et bevis for den opprinnelige ulikheten.
- I virkelighetens verden kan vi trenge mer komplekse unifiseringer når vi prøver å finne bevis for påstander, og det å organisere søket på en slik måte at vi ofte raskt finner bevis der de finnes er et viktig teknologisk aspekt.
- Med dette gir vi oss med unifisering.
- Dette avslutter også innføringen i grafer og trær.
- Det som gjenstår nå er å se på en rekke eksempeloppgaver rundt dette stoffet.

Forelesning 28

Grafer og trær, eksempler

Dag Normann - 5. mai 2008

Grafer og trær

- I dag skal vi se på en rekke eksempeloppgaver, og gjennomgå løsningene på tavla.
- Alle eksemplene er oppgaver som ville kunne bli gitt til eksamen, enten alene, eller som en del av en større oppgave.
- Eksemplene er hentet fra kapitlene om grafer og trær, og fra det stoffet i tilknytning til disse kapitlene som ikke er behandlet i læreboka.
- Selvfølgelig vil ikke eksamen kunne dekke alle aspekter ved grafer og trær.

Eksempel.

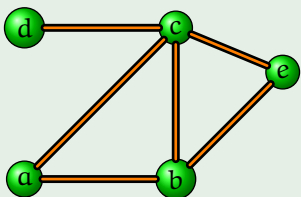
- a) Vi har gitt forslag til hva graden til nodene i en graf er.
Det er bare to av forslagene som kan realiseres av en graf.
Finn ut hvilke forslag som ikke kan realiseres, og forklar hvorfor ikke.
1. To noder med grad 1, to noder med grad 2 og en node med grad 3.
 2. To noder med grad 1, to noder med grad 2 og to noder med grad 3.
 3. En node med grad 3 og to noder med grad 4.
 4. To noder med grad 2, to noder med grad 3 og en node med grad 4.
- b) Finn eksempler på sammenhengende grafer som har noder og grader som beskrevet i de to tilfellene det er mulig.
- c) En av grafene vil ha en Eulersti. Finn en Eulersti i den grafen som har det, og forklar hvorfor den andre grafen ikke har noen Eulersti.

Eksempel.

- a) Hvis G er en graf med fem noder, hva er det minste antall kanter G kan ha og likevel være sammenhengende.
- b) Finn to sammenhengende grafer med minimalt antall noder som ikke er isomorfe. Forklar hvorfor de ikke er isomorfe.
- c) Finnes det to sammenhengende grafer med fem noder og minimalt med kanter som ikke er isomorfe, slik at begge har en Eulersti?
Begrunn svaret.

Eksempel.

- a) Vis hvordan følgende graf kan representeres som en matrise:



- b) Bestem graden til hver av nodene og avgjør om grafen har en Eulersti eller en Eulerkrets. Begrunn svaret ditt.

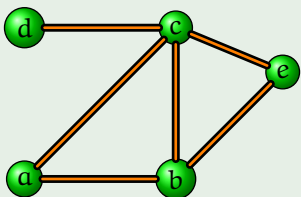
Eksempel.

- a) Forklar forskjellen på en krets og en sykel.

Er alle kretser sykler?

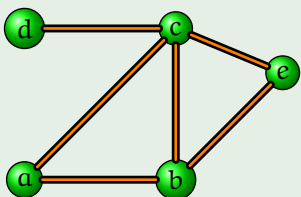
Er alle sykler kretser?

- b) Finn alle sykler i grafen under, og finn en krets som ikke er en sykel.



Eksempel.

- a) Finn komplementet til grafen



- b) Er komplementet sammenhengende?

Eksempel.

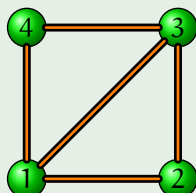
- a) Finnes det et tre med ni noder slik at fire av disse nodene har grad 4?

Begrunn svaret.

- b) Finn to trær som ikke er isomorfe slik at begge har ni noder hvorav to har grad 4.

Eksempel.

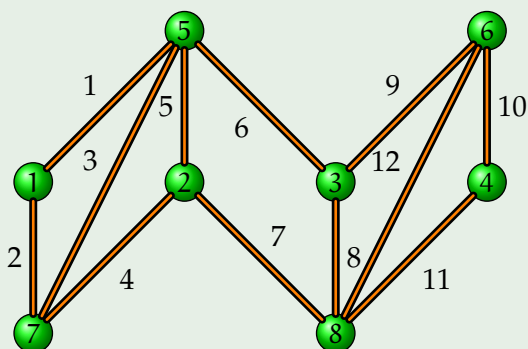
- a) Finn et utspennende tre i følgende graf:



- b) Finn et annet utspennende tre som ikke er isomorft med det første.
c) Kan det finnes utspennende trær som ikke er isomort med noen av de to andre?

Eksempel.

Vi skal knytte to oppgaver til følgende vektete graf:



Eksempel.

- a) Ved å bruke Prims algoritme, finn det minimale utspennende treet i den vektete grafen på forrige side.
Forklar hvor du begynner, og i hvilken rekkefølge du legger kanter til treet.
- b) Finn avstanden mellom node 7 og node 6 i grafen.
- c) Bruk Dijkstras algoritme til å finne treet som gir de korteste veiene fra alle noder til node 6
Forklar i hvilken rekkefølge du vil legge kanter til treet.

Eksempel.

Anta at vi utvider den vektete grafen fra side 10 med en kant fra node 5 til node 6 med vekt 7,5 og en kant fra node 7 til node 8 med vekt 5,5.

- Vil en eller begge av disse kantene inngå i det minimale utspennende treet i den nye grafen?
- Har vi gjort det enklere, vanskeligere eller like enkelt å finne en Eulersti?
- Hva er den maksimale vekten av en sti som ikke benytter samme kant to ganger i de to tilfellene?

Eksempel.

- Tegn syntakstreet til følgende utsagnslogiske uttrykk:

$$(((\neg p \vee q) \wedge r) \vee (p \wedge \neg r)) \wedge (\neg q \vee r)$$

- Skriv dette uttrykket med polsk notasjon.
- Vis hvordan du kan bruke trerekursjon til å omforme syntakstreet til et uttrykk A på svak normalform til syntakstreet til den svake normalformen til $\neg A$.

Eksempel.

Vi arbeider med språket for enkle termer over \mathbb{J} som består av konstantene 1, 0 og -1 og hvor vi bruker funksjonene \times og $+$.

- Ett av de tre ordene under er en term hvor vi har brukt polsk notasjon.

Finn ut hvilken term det er og skriv den med vanlig infiks notasjon:

1. $+ \times 0 - 1 \times \times + 111$

2. $+ \times 0 - 1 \times \times + 1110$

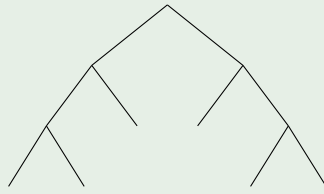
3. $+ \times 0 - 1 \times \times + 1110 - 1$

- Tegn forsøk på syntakstrær for de tre ordene, og foklar hvordan forsøkene avslører hvilke ord som ikke er akseptable.

Eksempel.

Vi arbeider med termer hvor vi bruker konstantene 0 og 1 og funksjonene $+$ og \times .

Et syntakstre har formen



- Hvor mange forskjellige termer kan dette være syntakstreet til?
- Sett merkelapper på treet slik at dette blir et ekte syntakstre og finn den tilsvarende termen på infiks og omvendt polsk form.
- Vis hvordan hver node kan markeres av en bit-sekvens (en 0 – 1 - sekvens).

Eksempel.

- Undersøk om følgende to termer

$$t_1 = (x \times (0 + y)) \times ((1 + 1) \times z)$$

og

$$t_2 = ((y + 0) \times (0 + 1)) \times ((y + y) \times (0 + 0))$$

lar seg unisere, det vil si, om vi kan finne termer for x , y og z slik at t_1 og t_2 blir like.

Eksempel.

Kan vi finne utsagnslogiske uttrykk A , B og C slik at følgende utsagn blir like?

–

$$(p \wedge q) \vee (A \wedge r) \vee (p \wedge r) \vee (A \wedge q)$$

–

$$((p \wedge q) \vee (\neg p \wedge C) \vee (p \wedge C) \vee B)$$

Forelesning 29

Kompleksitetsteori

Dag Normann - 7. mai 2008

KAPITTEL 13: Kompleksitetsteori

Meldinger:

Det blir hovedsaklig tavleregning på plenumsregningen i morgen.

15/5-08 vil D.N. vikariere for Roger på plenumsregningen.

Oppgavene til neste uke vil i hovedsak bestå av gamle eksamensoppgaver.

Kompleksitetsteori

- Vi skal nå starte på siste kapittel i boka, og dermed siste del av pensum.
- I denne siste delen skal vi spørre oss om det er mulig å måle hvor lang tid det tar å utføre en algoritme, og hva en slik måling innebærer.
- Det vil være interessant å vite om en gitt algoritme kan utføres av en datamaskin innen rimelig tid, og hvis vi har to algoritmer som skal løse den samme oppgaven, om den ene er raskere enn den andre.
- Før vi diskuterer hva disse spørsmålene kan bety, la oss se på et eksempel på dårlig og god programmering for å løse en oppgave.

Kompleksitetsteori

Eksempel.

- Sommeren 1986? fikk D.N. en telefon fra en person som ville starte et lotteri.
- Det skulle trykkes tre og en halv millioner lodd, med nummer fra 1 til 3.500.000.
- Tanken var at man skulle utbetale ekstragevinster for enkelte tverrsummer av loddnummeret.
- Tverrsummen vil være et tall mellom 1 og 56.
- ønsket var at en matematiker skulle hjelpe til med å finne ut hvor mange lodd som vil ha tverrsum i når i varierer fra 1 til 56.

Kompleksitetsteori

Eksempel (Fortsatt).

- Logikk-gruppen hadde to nyinnkjøpte Xerox arbeidsplassmaskiner, og første forsøk på å imøtekomme ønsket var å skrive et LISP-program (med hjelp fra stipendiat Erik Colban) som realiserte følgende algoritme:

```
1 For i = 1 to 56 do
  1.1  $x_i \leftarrow 0$ 
2 For n = 1 to 3.500.000 do
  2.1  $i \leftarrow \text{tverrsum}(n)$ 
  2.2  $x_i \leftarrow x_i + 1$ 
3 Output  $x_1, \dots, x_{56}$ .
```

Kompleksitetsteori

Eksempel (Fortsatt).

- Etter en god lunsj ble det klart at dette ville ta hele dagen, og trolig natten med.
- Vi måtte finne en raskere algoritme.
- Vi laget en algoritme basert på en matematisk analyse av problemet (analysen tok ca. 10 minutter)
- Den nye algoritmen ga svaret i løpet av ca 2 sekunder.

Kompleksitetsteori

Eksempel (Fortsatt).

- Vi ser kort på tankegangen bak den nye algoritmen:
- Vi kjenner til hvordan tverrsummene fordeler seg for tall fra 0 til 9.
- Tverrsummene for tall fra 10 til 19 fordeler seg nesten likt, bare forskjøvet et tall oppover.
- Tilsvarende kan vi lett finne fordelingen av tverrsummene for tall mellom 20 og 39, mellom 30 og 39 osv.
- Legger vi sammen får vi fordelingen av tverrsummer for alle tall mellom 0 og 99.

Kompleksitetsteori

Eksempel (Fortsatt).

- Forskyver vi denne fordelingen ett tall opp, får vi fordelingen av tverrsummer mellom 100 og 199.

- ti nye runder gir oss altså fordelingen av tverrsummene for alle tall mellom 0 og 999.
- Ti nye runder gir oss fordelingen opp til 9.999, osv. Tilslutt må vi justere tallene litt slik at vi får fordelingen av tverrsummer for tall mellom 1 og 3.500.000.
- Som en generell algoritme ser vi at antall regneoperasjoner er proporsjonalt med antall sifre i antall lodd, og ikke i antall lodd.
- Vi skal komme tilbake til slike fenomener senere, og sette navn på dem. Forklaringen er justert i forhold til det som ble sagt på forelesningen.

Kompleksitetsteori

- I kompleksitetsteori er det ofte to størrelser man prøver å finne
 - Hvor lang tid tar det å følge en algoritme.
 - Hvor mye lagerplass må man sette av for at algoritmen skal ha den informasjonen den trenger til enhver tid.
- Vi skal konsentrere oss om tidskompleksitet, *time complexity* og la plasskompleksitet, *space complexity* være udiskutert.

Kompleksitetsteori

- Vi skal ikke ta sikte på å gi en innføring i kompleksitetsteori som en del av den teoretiske informatikken, men at dere etter endt MAT1030 kan
 - a) Vurdere to algoritmer mot hverandre for å kunne vurdere hvilken som vil være mest tidseffektiv.
 - b) Vurdere om en algoritme er gjennomførbar innen akseptabel tid for input av den størrelsen man ønsker at den skal virke for.

Kompleksitetsteori

Hvis vi har skrevet en algoritme på pseudokodeform, er spørsmålet om hvor lang tid det tar å følge algoritmen et upresist spørsmål av flere grunner:

- Svaret avhenger av hvilket programmeringsspråk vi benytter.
- Svaret avhenger av hvilken maskin vi benytter.
- Svaret kan avhenge av hvor stort minne vi har satt av til hjelp for programmet.
- Svaret kan avhenge av om vi må dele ressursene med andre.
- Svaret vil avhenge av input, og av hvordan input representeres digitalt.
- med mer, med mer.

Vi trenger imidlertid ofte ikke å kjenne alle disse forholdene for å kunne sammenlikne algoritmer eller for å vurdere om en algoritme er praktisk gjennomførbar eller ikke.

Vi skal lære å se bort fra det uvesentlige, og derigjennom få et grunnlag for å vurdere den omtrentlige kompleksiteten av en algoritme.

Det vil være den omtrentlige tidsbruken som funksjon av størrelsen på input vi vil være på jakt etter.

Kompleksitetsteori

Eksempel.

```
1 Input n [n naturlig tall]
2 x ← 1
3 For i = 2 to n do
    3.1 x ← x · i
4 y ← 0
5 For j = 1 to x do
    5.1 y ← y + j
6 Output y
```

Kompleksitetsteori

Eksempel (Fortsatt).

- Vi har gitt en pseudokode for å beregne

$$f(n) = \sum_{j=1}^{n!} j.$$

- Vi må anta at det normalt krever mere tid å multiplisere to tall enn å summere dem.
- På den annen side skal vi utføre $n - 1$ multiplikasjoner i den første delen, mens vi skal utføre ca. $n!$ addisjoner i andre del.
- n skal ikke være så veldig stor før andre del av algoritmen tar vesentlig lengere tid å utføre enn første del.
- Det er i andre del at kompleksiteten ligger.

Kompleksitetsteori

Eksempel (Fortsatt).

```
1 Input n [n naturlig tall]
2 x ← 1
3 For i = 2 to n do
    3.1 x ← x · i
4 y ←  $\frac{x(x-1)}{2}$ 
5 Output y
```

Kompleksitetsteori

Eksempel (Fortsatt).

- Den nye pseudokoden gir oss nøyaktig den samme funksjonen.
- Her vil fortsatt den første delen innebære at vi må foreta $n - 1$ multiplikasjoner, mens den andre delen innebærer essensielt en multiplikasjon og en divisjon med 2.
- Nå er det den første delen som vil være mest tidkrevende.
- Vi har funnet en raskere algoritme for å beregne den samme funksjonen.

Kompleksitetsteori

- Lærebokas første tilnærming til kompleksiteten av en algoritme lyder, oversatt til norsk:
Tell bare de mest tidkrevende operasjonene.
- En operasjon kan være en enkel regneoperasjon, en **for** - løkke, en sammenlikning av størrelser, en annen form for løkke eller noe annet.
- Hvis løkker inngår i algoritmen, vil ofte lengden på løkkene bestemme hvor tidkrevende algoritmen er.
- Det kan derfor være lurt, slik vi gjorde i eksemplet, å studere lengden på de enkelte løkkene.
- Mange addisjoner kan overskygge langt færre multiplikasjoner, selv om det er mer tidkrevende å utføre en multiplikasjon enn en addisjon.
- Hvis koden inneholder **while** - løkker eller **until** - løkker, kan det være vanskelig å sammenlikne tidsbruken med tidsbruken til andre løkker.

Kompleksitetsteori

- La oss se på Prims algoritme i lys av første tilnærming.
- I Prims algoritme har vi listet opp nodene i en vektet graf, og så har vi listet opp kantene i grafen sammen med sine vekter.
- I Prims algoritme har vi en hovedløkke hvor vi i løpet av løkka legger en ny kant til det utspendte treet.
- I skritt nr. i skal vi ta for oss hver av de $n - i$ nodene som ikke har kommet med i treet, se på alle kantene fra disse nodene til treet bygget så langt og plukke ut den av disse kantene som har minst vekt.
- Den mest tidkrevende enkeltoperasjonen vil være å vurdere om en kant er kandidat til å bli lagt til treet, samt å sammenlikne vekten av hver enkelt kant med vekten til en tidligere utplukket kandidat.
- Vi skal senere komme med en måte å formulere omtrent hvor mange slike grunnoperasjoner vi må utføre.

Forelesning 30

Kompleksitetsteori

Dag Normann - 14. mai 2008

Informasjon

- Det er lagt ut program for orakeltjenestene i MAT1030 denne våren på semestersiden.
- Det blir ikke ordinære gruppetimer fra og med neste uke.
- Oppgaveregningen i morgen blir ren tavleregning, ettersom Roger er bortreist, uten tilgang til epost, og vikaren har ikke tilgang til styringsfilen for oppgavefoilene.

Oppsummering

- Sist onsdag startet vi på kapitlet om kompleksitetsteori.
- Vi er interessert i å kunne si noe om hvor lang tid det tar å følge en algoritme.
- Målet er at vi skal kunne sammenlikne tidsbruken til forskjellige algoritmer, for å vurdere hvilken som er mest tidseffektiv.
- I tillegg skal vi kunne vurdere hvorvidt et program basert på en algoritme kan forventes å terminere for de ønskede input innen akseptabel tid.

- Kompleksitetsteori er en presis matematisk disiplin, men vi skal ikke drive den så langt.
- Vi vil følge boka, og finne frem til fire aspekter vi kan se på når vi skal vurdere effektiviteten av en algoritme.
- Det første aspektet var at vi skal konsentrere oss om de delene av algoritmen som tar lengst tid.
- Dette kan innebære å se på hvilke enkeltoperasjoner, hvor vi setter en verdi på en variabel, det er som er mest tidkrevende.
- Det viktigste er imidlertid å se på de forskjellige løkkene som skal gjennomkjøres under utførelsen av algoritmen, og å se på hvor mange regnetrinn de består av.
- Vi sammenfattet denne tilnærmingen med

Tell bare de mest tidkrevende operasjonene.

Det norske ordet "tilnærming" er normalt en grei oversettelse av det engelske "approximation", men det vil gi en riktigere intuisjon om vi erstatter det med forenkling.

Målet med disse tilnærmingene er at det skal bli mulig å sammenlikne algoritmer, og da viser det seg at det er enkelte forenklinger som gir det mest nyttige bildet.

Hvis vi oppfatter ordet tilnærming slik at det står for en tilnærmet beskrivelse av kompleksiteten til en algoritme, er dette noenlunde dekkende.

La oss så fortsette utforskningen av kompleksitetsteoriens verden.

Kompleksitetsteori

Eksempel.

```
1 Input n [n naturlig tall]
2 Input  $x_{n-1}, \dots, x_1$  [Hver  $x_i$  lik 0 eller 1]
3  $x_n \leftarrow 0$ 
4  $i \leftarrow 1$ 
5 While  $x_i = 1$  do
    5.1  $x_i \leftarrow 0$ 
    5.2  $i \leftarrow i + 1$ 
6  $x_i \leftarrow 1$ 
7 Output  $x_n \cdots x_1$ 
```

Eksempel (Fortsatt).

- Denne pseudokoden gir en algoritme for å legge 1 til det binære tallet $x_n \cdots x_1$.
- Hvis vi starter med $n = 20$ og det binære tallet 11111111111111111111, vil **while**-løkka gjentas nitten ganger, og vi tester om den skal brukes 20 ganger.
- Hvis vi starter med det binære tallet 111111111111111110 utfører vi testen for **while**-løkka bare en gang.
- Siden den eneste kontrollen vi har over hvor mange ganger denne løkka må gjentas er antall sifre i det binære tallet, lar vi det være målet på hvor lang tid vi bruker.

For endel algoritmer vil tiden vi bruker kunne avhenge av om vi er heldige med valg av input eller ikke.

Når vi skal vurdere kompleksiteten til en algoritme, kan det ofte være hensiktsmessig å vurdere tidsbruken i de verste tilfellene.

Det er dette læreboka setter opp som tilnærming nr 2, etter at man har vurdert hvilken del av programmet det er som overskygger de andre delene i tidsbruk:

*Hvis tidsbruken varierer for forskjellige input av samme størrelse,
ta utgangspunkt i det verste tilfellet.*

Eksempel.

- Vi har gitt en sammenhengende graf og skal avgjøre om grafen har en Eulerkrets eller ikke.
- Når vi skal vurdere kompleksiteten av en algoritme, er det viktig hvordan vi representerer input.

- Her vil vi anta at grafen er gitt som en symmetrisk matrise, hvor tallet i rad i og kolonne j angir hvor mange kanter det er mellom nodene i og j .
- Tallene på diagonalen skal være det dobbelte av antall løkker ved den tilsvarende noden.
- Graden til en node er da summen av alle tallene langs tilsvarende rad (eller søyle).

Eksempel (Fortsatt).

- Vi bestemmer om grafen har en Eulerkrets ved å summere tallene i hver rad til vi finner et oddetall.
- Har grafen en Eulerkrets, må vi summere tallene i alle radene, så hvis n er antall noder, må vi utføre $n(n - 1)$ addisjoner og sjekke at n tall er partall.
- Hvis grafen ikke har en Eulerkrets kan vi slippe billig fra det og utføre bare $n - 1$ addisjoner.
- Den dominerende prosessen i det verste tilfellet er det å summere tallene i alle radene, så det er de operasjonene vi legger til grunn når vi vurderer kompleksiteten.

Eksempel (Fortsatt).

- Anta nå at vi ikke visste at grafen var sammenhengende.
- Er det ødeleggende for kompleksiteten av problemet hvorvidt grafen har en Eulerkrets at vi må undersøke om den er sammenhengende?
- Vi kan uformelt beskrive en prosedyre som undersøker om en graf er sammenhengende på følgende måte:
 - Vi vil finne sammenhengskomponenten til node 1:
 - La A være $n \times n$ -matrisen til G hvor $a_{i,j}$ er tallet i rad i og søyle j .
 - La $X_1 = \{1\}$
 - Ved rekursjon for $k < n$, la $X_{k+1} = \{j \leq n \mid \exists i \in X_k (a_{i,j} > 0)\}$.
 - G er sammenhengende hvis $X_n = \{1, \dots, n\}$.

Eksempel (Fortsatt).

- I denne algoritmen har vi en hovedløkke i n trinn.
- Hvert trinn i løkka består av en gjennomløpning av alle par av noder, for å se om det finnes en kant som forbinder den ene noden med sammenhengskomponenten bygget opp så langt.
- Det å undersøke om en graf er sammenhengende krever altså flere operasjoner enn det å undersøke om den har en Eulerkrets, når vi gjør det på denne måten.

Eksempel.

- Det neste eksemplet som skal belyse tilnærming 2 er Euklids algoritme.
- Euklids algoritme er en selvkallende algoritme som finner det største felles mål for to tall.
- Det største felles målet er det samme som den største felles faktoren.
- Hvis $n \geq m$ er to naturlige tall vil $\text{Euklid}(n, m)$ være
 - m hvis m er en faktor i n .
 - $\text{Euklid}(m, k)$ hvor k er resten når vi deler n på m når m ikke er en faktor i n .
- Euklids algoritme er rask, selv for store tall.

Eksempel (Fortsatt).

- Hvis vi følger Euklids algoritme for to tallpar som ligger nær hverandre ser vi at det likevel kan være forskjeller i hvor raskt algoritmen gir et svar.
 1. $(80, 32) \rightarrow (32, 16)$ som gir svar 16.
 2. $(81, 32) \rightarrow (32, 17) \rightarrow (17, 15) \rightarrow (15, 2) \rightarrow (2, 1)$ som gir svaret 1
- Hvordan skal vi så kunne finne de verste tilfellene?
- Følg med på den overraskende fortsettelsen!

Eksempel (Fortsatt).

- Det minste par av forskjellige tall som gir oss svaret med en gang er $(2, 1)$
- Det minste tallet > 2 som gir 1 som rest når vi deler det med 2 er $1 + 2 = 3$
- Det minste tallet > 3 som gir 2 som rest når vi deler det med 3 er $3 + 2 = 5$.
- Det minste tallet > 5 som gir 3 som rest når vi deler det med 5 er $5 + 3 = 8$

Eksempel (Fortsatt).

- Hvis vi begynner med et par av Fibonaccitall (F_{n+1}, F_n) vil Euklids algoritme gi oss parene (F_n, F_{n-1}) i neste omgang.
- Dette er de verste tilfellene, det vil si de tilfellene hvor vi bruker lengst tid i forhold til hvor store tallene er.
- Dette var neppe en anvendelse Fibonacci hadde i tankene, men hvem vet?

- Når vi skal vurdere om en algoritme er raskere enn en annen, er det ikke sikkert at det er relevant for alle input.
- Det kan lønne seg å benytte en algoritme som arbeider raskere for store input, der tiden vi bruker faktisk kan ha økonomisk betydning, selv om en annen algoritme er bedre for små input.
- Vi skal først illustrere dette ved å gå gjennom et eksempel i boka, ettersom dette eksemplet i seg selv er viktig.
- Det dreier seg om effektiv eksponensiering, det vil si, om en metode for raskt å kunne beregne store potenser av et tall.
- Eksemplet har samme verdi om vi regner potenser av reelle tall, naturlige tall eller hele tall, så det presiserer vi ikke.

Eksempel.

- Vi kan definere funksjonen $f(x, n) = x^n$ ved rekursjon som følger:
 - $x^0 = 1$
 - $x^{n+1} = x^n \cdot x$
- Skal vi bruke denne til å beregne 3^8 får vi følgende beregning:

Eksempel (Fortsatt).

1. $3^0 = 1$
2. $3^1 = 1 \cdot 3 = 3$
3. $3^2 = 3 \cdot 3 = 9$
4. $3^3 = 9 \cdot 3 = 27$
5. $3^4 = 27 \cdot 3 = 81$
6. $3^5 = 81 \cdot 3 = 243$
7. $3^6 = 243 \cdot 3 = 729$
8. $3^7 = 729 \cdot 3 = 2187$
9. $3^8 = 2187 \cdot 3 = 6561$

Eksempel (Fortsatt).

- Som mennesker utfører vi de første multiplikasjonene raskere enn de siste, men for en maskin som arbeider med fullstendige binære representasjoner er en multiplikasjon en multiplikasjon, og tar omtrent like lang tid uansett hvordan faktorene ser ut.
- I realiteten må vi utføre seks multiplikasjoner for å beregne 3^8 på denne måten.

Eksempel (Fortsatt).

- En alternativ måte å beregne 3^8 på kan være:
 1. $3^2 = 3 \cdot 3 = 9$
 2. $3^4 = 3^2 \cdot 3^2 = 9 \cdot 9 = 81$
 3. $3^8 = 3^4 \cdot 3^4 = 81 \cdot 81 = 6561$
- Her bruker vi bare tre multiplikasjoner i motsetning til seks.
- Skulle vi beregnet 3^{16} ville vi etter den første metoden måtte utføre 8 nye multiplikasjoner, mens vi etter den nye metoden klarer oss med en til:
$$3^{16} = 3^8 \cdot 3^8 = 6561 \cdot 6561 = 43046721$$
- Dette går faktisk forttere, selv for hånd.
(Eller gjør det det?)

Med utgangspunkt i siste eksempel, skal vi nå beskrive to algoritmer for eksponensiering, og sammenlikne dem.

Vi har sett på hvordan vi kan beregne x^1, x^2, x^4, x^8 og så videre ved gjentatt kvadrering. Hvordan skal vi for eksempel kunne utnytte dette til å beregne x^{13} ?

Vi vet at $x^{13} = x^8 \cdot x^4 \cdot x$

Vi vet at 13, representert som binært tall, er 1101_2

En strategi kan derfor være at vi beregner x, x^2, x^4 og x^8 samtidig som vi ser på binærrepresentasjonen av 13 for å se hvilke av disse tallene som skal inngå som et produkt i x^{13} .

Siden 13 faktisk er gitt ved sin binære representasjon i en datamaskin, er dette veldig gunstig.

Vi skal gi en fullstendig pseudokode for å beregne x^n når n er gitt på binær form, men først skal vi se på et eksempel:

Eksempel.

- Vi vil beregne 3^{22}
- $22 = 16 + 4 + 2$ så binærformen til 22 er 10110
- Vi vil beregne to følger:
 1. Den ene er $3, 3^2, 3^4, 3^8$ og 3^{16} slik vi har sett før.
 2. Den andre er produktet av de tallene i den første følgen som inngår i 3^{22} etterhvert som vi kommer til dem.
- Vi ser på hvilke tallpar vi får underveis, og hvordan vi kommer frem til dem:
 1. $y_1 = 3 = 3^{2^1}$ og $z_1 = 1$ fordi siste siffer i 10110 er 0.
 2. $y_2 = 3 \cdot 3 = 9$ og $z_2 = 9 \cdot 1 = 9$
 3. $y_3 = 9 \cdot 9 = 81$ og $z_3 = 81 \cdot 9 = 279$

4. $y_4 = 81 \cdot 81 = 6561$ og $z_4 = 279$

5. $y_5 = 6561 \cdot 6561 = 43046721$ og $z_5 = 43046721 \cdot 279 = 12010035159$

- Svaret er 12010035159.

```
1 Input x [x et reelt tall]
2 Input k [k antall sifre i binærrepresentasjonen av n]
3 Input  $b_k \cdots b_1$  [Binærrepresentasjonen av n]
4  $y \leftarrow x$ 
5  $z \leftarrow 1$ 
6 For i = 1 to k do
    6.1 If  $b_i = 1$  then
        6.1.1  $z \leftarrow y \cdot z$ 
    6.2  $y \leftarrow y \cdot y$ 
7 Output z
```

- Denne pseudokoden er litt anderledes enn den som står i boka.
- Skal vi beregne x^2 tar denne prosedyren litt mer tid enn den definert ved rekursjon, ettersom vi her får både å regne ut x^2 og $x^2 \cdot 1$, men for store n er denne algoritmen vesentlig raskere.

Eksempel.

```
1 Input n [n naturlig tall]
2  $x \leftarrow 0$ 
3 For i = 1 to n do
    3.1  $x \leftarrow 2x$ 
4 Output x
```

Det vi gjør her er å regne ut $x = 0$ ved rekursivt å multiplisere 0 med 2^n .

Vi kan finne en annen algoritme som beregner den samme funksjonen:

Eksempel (Fortsatt).

```
1 Input n
2  $x \leftarrow \frac{3 \cdot 5 - 15}{n \cdot (n+1)}$ 
3 Output x.
```

I det siste eksemplet må vi foreta fem regneoperasjoner, mens i det første eksemplet er antall regneoperasjoner avhengig av n .

For små n vil den første algoritmen faktisk gi raskere svar, også fordi vi der kan arbeide med hele tall, mens vi må arbeide med flytende reelle tall i den andre algoritmen.

For store input er imidlertid den andre, direkte metoden raskere enn den første.

Ved å følge tredje tilnærming, stopper all diskusjon om hvilken av to dumme algoritmer som er best.

- Hvis input er lite, vil de fleste algoritmer gi oss et svar innen rimelig tid, og det spiller ikke så stor rolle hvilken algoritme vi velger hvis det er flere mulige.
- Hvis input er stort, kan en ineffektiv algoritme bruke ødeleggende mye mer tid enn en effektiv algoritme.
- Det er derfor at tidsbruken for store inputverdier er det mest interessante.
- Dette er samlet i tredje tilnærming

Anta at input er stort

- Vi har sammenliknet algoritmer, og vi har drøftet kompleksitet i visse tilfeller, men vi har ikke sagt så mye om hva slags funksjoner vi vil bruke til å måle kompleksitet med.
- Data er gitt på digital form, og det er naturlig å måle størrelsen på input ut fra hvor mange bits som brukes til å representere input.
- La oss gå tilbake til eksemplet om grafer og problemet om å avgjøre om en graf er sammenhengende eller ikke.
- Siden løkker og parallelle kanter ikke kan gjøre en graf mer sammenhengende, kan vi godt begrense dette problemet til enkle grafer, det vil si grafer uten løkker og parallelle kanter.
- Uten å gå i detalj, kan vi si at for å representere en enkel graf med n noder, trenger vi et antall bits begrenset av $k \cdot n^2$ hvor k er et tall uavhengig av n men avhengig av hvordan vi velger å representere grafen digitalt.
- Snur vi dette, ser vi at hvis m er antall bits i input, er antall noder i grafen begrenset av et tall $a \cdot \sqrt{m}$ hvor a er en konstant uavhengig av m .
- Da vi lagde en prosedyre for å bestemme om en graf med n noder er sammenhengende eller ikke, forestilte vi oss en prosess i følgende trinn:
 1. Velg ut en node.
 2. I $n - 1$ runder, utvid noden til en maksimal sammenhengende delgraf, ved i hvert trinn å legge til de nye nodene som kan nås fra delgrafens bygget opp så langt ved å legge til en kant.
 3. Undersøk om det finnes noder som ikke er med i sammenhengskomponenten.
- I hvert skritt i hovedløkka, gikk vi gjennom alle kantene, for å se om en av endenodene lå i grafen konstruert så langt.
- Hvis input er på m bits, har vi ca. $m^{\frac{1}{2}}$ trinn i hovedløkka og vi må (i verste tilfelle) teste ca. $\frac{1}{2} \cdot m$ kanter.
- Siden vi opererer med cirkatall, vi skal se på de verste tilfellene og bare på den mest tidkrevende delen av algoritmen, får vi at tidsbruken er omtrent $m^{\frac{3}{2}}$ hvor m er antall bits i input.

- Vi skal etterhvert være litt mer presis i hva vi mener med “cirka”.

Definisjon.

En polynomfunksjon er en funksjon på formen

$$f(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$$

Vi antar normalt at $a_k \neq 0$, og da er k graden til funksjonen.

- I noen tilfeller er det viktig å skille mellom polynomfunksjonen og polynomet, som er det definerende uttrykket.
- Dette er ikke viktig for oss.
- Hvis graden til en polynomfunksjon f er større enn graden til en annen funksjon g , vil $f(n) > g(n)$ bare n er stor nok.
- Det betyr at hvis kompleksiteten til to algoritmer er gitt ved polynomfunksjoner, kan vi bruke tilnærming 3 og bestemme hvilken som er den raskeste hvis gradene er forskjellige.

Eksempel.

- Vi har gitt et stort tall på binær form og vil undersøke om tallet er et av Fibonacci-tallene.
- Det gitte tallet er representert ved n bits.
- Vi setter av fire n -bits områder R_1 , R_2 , R_3 og R_4 hvor det gitte tallet ligger i R_1 .
- Vi starter med å laste binærkoden til 1 i R_2 og binærrepresentasjonen til 2 i R_3
- Dette tar $n + n$ enkeltoperasjoner (siden vi må rydde R_2 og R_3 for søppel).

Eksempel (Fortsatt).

- Deretter starter vi en løkke hvor vi
 1. Laster summen av tallene i R_2 og R_3 inn i R_4 . Dette tar ca $2n$ regneskritt, siden vi må holde orden på eventuell mente.
 2. Sammenlikner verdien av R_1 og R_4 . Er de like, svarer vi JA, er tallet i R_4 størst, svarer vi NEI og er tallet i R_1 fortsatt størst, fortsetter vi prosessen.
 3. Laster tallet i R_3 over i R_2 og deretter tallet i R_4 over i R_3 . Dette tar ca $2n$ regneskritt.
- Antall ganger vi må gjennomføre denne løkka er tilnærmet proporsjonal med n etter som Fibonacci-tallene øker tilnærmet eksponensielt.
- Det betyr at vi kan bruke en annengradsfunksjon til å beskrive den omtrentlige tidsbruken, $a \cdot n$ løkker som hver bruker ca $b \cdot n$ regneskritt.

- I det forrige eksemplet så vi at hvis m er et tall gitt på binær form med n siffre, finnes det en konstant c slik at antall regneskritt som skal til for å avgjøre om m er et Fibonacci-tall eller ikke er begrenset av

$$f(n) = c \cdot n^2.$$

- Vi var ikke spesielt ivrige etter å finne en konkret verdi på c , av forskjellige grunner:
 1. c vil avhenge av hvilket språk vi bruker og faktisk av hvilken maskin vi bruker.
 2. Den virkelige tiden avhenger vel så mye av hvor kraftig maskinvare vi disponerer som hvor liten vi kan få verdien på c til å bli.
 3. Den teknologiske utviklingen gjør at selv store verdier for c er uten betydning for effekten av denne algoritmen.
- Det som ville hjulpet var om vi kunne bringe kompleksiteten ned fra, si $40 \cdot n^2$ til $1.000 \cdot n$.

Forelesning 31

Kompleksitetsteori

Dag Normann - 19. mai 2008

Informasjon

Jeg er blitt bedt om å opplyse om hvilke forelesninger det er som inneholder eksamens-relevant stoff som ikke står i læreboka.

Det er

- Forelesning 17, 10. mars.
- Forelesning 18, 12. mars.
- Forelesning 26, 28. april.
- Forelesning 27, 30. april.
- Forelesning 28, 5. mai.

Kompleksitetsteori

- Vi nærmer oss slutten, selv på avsnittet om kompleksitetsteori.
- Vi har vurdert tre forenklinger, eller tilnærminger, vi kan gjøre når vi skal vurdere tidskompleksiteten av en algoritme
 1. Vurder den mest tidkrevende operasjonen.
 2. Betrakt alltid det verste tilfellet når flere input kan ha samme størrelse.
 3. Anta at input er stort.
- I noen eksempler har vi sett på hvor lange løkker vi kan ha, om løkker inneholder underløkker (i tilfellet hvor vi undersøker om en graf er sammenhengende), og vi har funnet frem til en størrelsesorden på kompleksiteten, som $f(n) = n^2$ eller $g(n) = n^3$.
- Vi kan for eksempel konkludere med at regnetiden vil være begrenset av $c \cdot n^2$ for input av størrelse n , hvor vi ikke bryr oss om å finne verdien på c .
- Det er flere grunner til at vi ikke vil bryr oss om å finne verdien på c , tre av de viktigste er:
 1. Verdien på c avhenger av programmeringsspråk, maskin og andre varierende forhold.
 2. Selv i en konkret situasjon kan det være vanskelig å bestemme en fornuftig verdi av c .
 3. Det har liten teknologisk interesse i de fleste tilfellene hva verdien på c er.
- Alle disse betraktningene leder opp mot den fjerde tilnærmingen vi skal gjøre, og til det begrepet vi skal innføre.
- Fjerde tilnærming lyder:

Vi skiller ikke mellom to tidskompleksiteter hvis vekstraten til den ene er et konstant multiplum av vekstraten til den andre.

- Etter at vi nå har innført fire prinsipper for tilnærminger, hvorav tre av dem er mer å betrakte som tommelfingerregler enn matematisk presise regler, skal vi innføre den såkalte O-notasjonen (ikke “null”, men bokstaven O).
- Ved hjelp av den blir faktisk bruk av første, tredje og fjerde regel presise.
- Den vil også gjøre det mer presist å avgjøre hva som faktisk er de verste tilfellene.

Definisjon.

La f og g være tidskompleksiteter, det vil si, funksjoner fra \mathbb{N} til \mathbb{N} .

Vi sier at f er $O(g)$ hvis det finnes en positiv konstant c slik at

$$f(n) \leq c \cdot g(n)$$

for alle tilstrekkelig store n .

- Med “tilstrekkelig store” mener vi at det finnes en n_0 slik at ulikheten holder for alle $n \geq n_0$.
- Skulle vi gitt denne definisjonen mer presist, måtte vi bruke kvantorene vi lærte om tidligere i semesteret.
- Da ser definisjonen av at f er $O(g)$ slik ut:

$$\exists c > 0 \exists n_0 \forall n \geq n_0 (f(n) \leq c \cdot g(n)).$$

- Med denne notasjonen, og i lys av et eksempel vi har sett på før, kan vi si at tidskompleksiteten for den naturlige algoritmen som undersøker om en graf er sammenhengende og har en Eulerkrets er $O(n^{\frac{3}{2}})$, når n er antall bits vi trenger for å representere grafen.
- I motsetning til tidligere formuleringer som “størrelsesorden er...”, er dette et presist matematisk utsagn, og dekker alle reelle implementeringer av algoritmen vår.
- Bruken av denne notasjonen er så viktig at vi skal spandere på oss endel eksempler for å få litt intuisjon rundt den.
- Vi minner om at en polynomfunksjon er en funksjon

$$f(n) = a_k n^k + \dots + a_1 n + a_0.$$

- Vi skal anta at alle koeffisientene er i \mathbb{N}_0 , det vil si ikkenegative hele tall.
- Videre vil vi normalt anta at $a_k > 0$, og polynomfunksjonen har da grad k .
- Vi skal se på sammenhengen mellom O-notasjonen og polynomfunksjoner.

Eksempel.

La $f(n) = 3n + 2$ og $g(n) = 2n$.

Da er $f \in O(g)$ fordi $f(n) \leq 2g(n)$ når $n \geq 2$.

Eksempel.

La $f(n) = 10^6 \cdot n$ og la $g(n) = n^2$.

Er $f \in O(g)$?

Vi kan lett finne en verdi av c som viser dette veldig enkelt:

$f(n) \leq 10^6 \cdot g(n)$ for alle n .

Vi har også at $f(n) \leq g(n)$ for alle $n \geq 10^6$.

Eksempel.

La $f(n) = n^2$ og la $g(n) = 10^6 \cdot n$.

Er $f \in O(g)$?

I dette tilfellet er svaret negativt.

For å vise det, må vi vise at det ikke finnes noen c som duger.

For å vise at c ikke duger, må vi vise at det finnes vilkårlig store n slik at $c \cdot g(n) < f(n)$.

La $n > 10^6 \cdot c$.

Da er $f(n) = n^2 > 10^6 \cdot c \cdot n = c \cdot g(n)$.

Dette viser at svaret er negativt.

Eksempel.

La $f(n) = 3n^4 + 10n^3 + 2n + 20$ og la $g(n) = n^4$.

Er $f \in O(g)$?

Svaret er *JA*, og vi skal gi et argument som er så generelt at det tjener som bevis for neste teorem.

La $c = 3 + 10 + 2 + 20 = 35$, det vil si, summen av alle koeffisientene i f .

Husk at $n \geq 1$ her.

Da er

$$f(n) = 3n^4 + 10n^3 + 2n + 20 \leq$$

$$3n^4 + 10n^4 + 2n^4 + 20n^4 =$$

$$(3 + 10 + 2 + 20)n^4 = c \cdot g(n)$$

Denne metoden kan brukes til å vise følgende teorem.

Teorem.

Hvis f er en polynomfunksjon med grad $\leq k$ vil f være $O(n^k)$.

- Hva hvis graden til f er større enn graden til g ?
- Vi har sett et eksempel på dette hvor f ikke er $O(g)$.
- Det gjelder helt generelt, og vi skal se på et eksempel som illustrerer det.

Eksempel.

La $f(n) = n^3$ og la $g(n) = 2n^2 + 4n + 6$.

La c være en vilkårlig positiv konstant.

Vi vil vise at det finnes vilkårlig store n slik at $c \cdot g(n) < f(n)$.

Velger vi $n > (2 + 4 + 6)c = 12c$ får vi

$$\begin{aligned} f(n) &= n^3 > c \cdot (2 + 4 + 6)n^2 \\ &\geq c \cdot g(n) \text{ (som i beviset for teoremet).} \end{aligned}$$

Vi kan oppsummere dette med følgende observasjon:

Korollar.

- a) Hvis f og g er to polynomfunksjoner og f er $O(g)$, vil graden til f være mindre eller lik graden til g .
- b) Omvendt, hvis f og g er to polynomfunksjoner slik at graden til f er mindre eller lik graden til g vil f være $O(g)$.

- Vi har definert relasjonen

$$f \text{ er } O(g)$$

og det ville vært dumt å ikke benytte anledningen til å repetere litt om relasjoner i denne forbindelse.

- Vi husker at en relasjon R er transitiv hvis

$$aRb \wedge bRc \Rightarrow aRc.$$

- Er O -notasjonstrelasjonen transitiv?
- La oss drive litt undersøkende matematikk og anta at f er $O(g)$ og at g er $O(h)$.

- Da finnes det $c > 0$ og n_0 slik at hvis $n \geq n_0$ vil

$$f(n) \leq c \cdot g(n).$$

- Videre finnes det $d > 0$ og n_1 slik at hvis $n \geq n_1$ vil

$$g(n) \geq d \cdot h(n).$$

- Hvis vi nå lar $n \geq \max\{n_0, n_1\}$ har vi at

$$f(n) \geq c \cdot g(n) \geq c \cdot d \cdot h(n),$$

så konstanten $c \cdot d > 0$ kan brukes til å vise at f er $O(h)$.

- Dette viser at relasjonen er transitiv.
- Vi husker også at en relasjon R kalles refleksiv hvis aRa for alle a i grunnmengden.
- Er relasjonen

$$f \text{ er } O(g)$$

refleksiv?

- For alle funksjoner f og for alle tall n er $f(n) \leq 1 \cdot f(n)$, så f er $O(f)$ for alle f .
- Det viser at relasjonen er refleksiv.
- På generelt grunnlag kan vi da definere relasjonen f og g har samme kompleksitet ved f er $O(g)$ og g er $O(f)$.
- Siden vi tar utgangspunkt i en relasjon som er transitiv og refleksiv, får vi en ekvivalensrelasjon på denne måten.
- Ekvivalensklassene til denne relasjonen kaller vi ofte kompleksitetsklasser og de svarer til mengder av funksjoner hvor alle har samme kompleksitet ut fra forenklingene 1, 3 og 4.
- Dette er et eksempel på hvordan man kan bruke teorien for relasjoner til å gjøre et upresist begrep "vokser omtrent like fort" til et presist begrep.
- To polynomfunksjoner tilhører samme ekvivalensklasse nøyaktig når graden er den samme.
- Med dette avslutter vi innføringen i O -notasjonen.

Sortering

- Hvis man skal kunne skaffe seg oversikt over informasjonen i en stor datamengde, er det viktig å kunne sortere disse dataene etter visse kriterier.
- Det finnes en elektronisk tabell over ca. 10.000 vitenskapelige tidsskrift som ansatte ved norske universiteter og høyskoler har anledning til å publisere artikler i.
- Denne tabellen inneholder mye informasjon om det enkelte tidsskriftet, som ISSN-nummer, navn, fagområde, hvor mange artikler som er trykket der siste år, hvor mange artikler som er trykket der de siste fem årene, samt noen indekser som skal måle kvaliteten på tidsskriftet.

- For enkelte formål kan det være aktuelt å sortere disse ca. 10.000 tidsskriftene etter navn, for andre etter fagområder, og enkelte ganger er det mest formålstjenlig å sortere tidsskriftene etter et av kvalitetsmålene.
- Denne tabellen, og hvordan faglaugene forholder seg til informasjonen i den, har betydning for deler av finansieringen av universiteter og høyskoler.
- Det er derfor viktig at de som bruker denne tabellen til å formulere kriteriene for denne finansieringen raskt kan sortere dataene slik man har bruk for der og da.
- Det vi skal se på nå er en enkel metode for å sortere data, og på hvordan denne metoden kan forbedres slik at kompleksiteten bringes betraktelig ned.
- Begge de algoritmene vi skal se på illustrerer nytten av å forstå teorien rundt relasjoner.
- Vi vil bruke symbolet $<$ og forestille oss at vi arbeider med tall, men vi har bare bruk for at $<$ er en transitiv, irrefleksiv relasjon slik at vi for alle a og b har at $a = b$, $a < b$ eller $b < a$, men at vi ikke har flere muligheter.
- La oss se på et eksempel.
- Vi har gitt 10 tall uten noen spesiell orden, 5, 9, 4, 1, 7, 12, 3, 6, 2, 8.
- Vi vil sortere disse i voksende orden.
- Dette vil vi i første omgang gjøre i ti operasjoner.

Eksempel (Sortering av 5, 9, 4, 1, 7, 12, 3, 6, 2, 8).

1. Sorter 5 ; Rest 9, 4, 1, 7, 12, 3, 6, 2, 8
2. Sorter 5, 9 ; Rest 4, 1, 7, 12, 3, 6, 2, 8
3. Sorter 5, 9, 4 \rightarrow 5, 4, 9 \rightarrow 4, 5, 9 ; Rest 1, 7, 12, 3, 2, 8
4. Sorter 4, 5, 9, 1 \rightarrow 4, 5, 1, 9 \rightarrow 4, 1, 5, 9 \rightarrow 1, 4, 5, 9 ; Rest 7, 12, 3, 2, 8
5. Sorter 1, 4, 5, 9, 7 \rightarrow 1, 4, 5, 7, 9 ; Rest 12, 3, 2, 8
6. Sorter 1, 4, 5, 7, 9, 12 ; Rest 3, 2, 8
7. Sorter 1, 4, 5, 7, 9, 12, 3 $\rightarrow \dots \rightarrow$ 1, 4, 3, 5, 7, 9, 12 \rightarrow 1, 3, 4, 5, 7, 9, 12 ; Rest 2, 8
8. Sorter 1, 3, 4, 5, 7, 9, 12, 2 $\rightarrow \dots \rightarrow$ 1, 3, 2, 4, 5, 7, 8, 9, 12 \rightarrow 1, 2, 3, 4, 5, 7, 9, 12 ; Rest 8
9. Til sist flytter vi 8 nedover i den sorterte delen av listen til vi finner plassen dens, og den sorterte listen blir 1, 2, 3, 4, 5, 7, 8, 9, 12.

Her er sorteringsalgoritmen fra boka.

1 *Input* x_1, x_2, \dots, x_n

2 **For** $i = 2$ **to** n **do**

2.1 *plassér* $\leftarrow x_i$

2.2 $j \leftarrow i - 1$

2.3 **While** $j \geq 1$ *and* $x_j > \textit{plassér}$ **do**

2.3.1 $x_{j+1} \leftarrow x_j$

2.3.2 $j \leftarrow j - 1$

2.4 $x_{j+1} \leftarrow \text{plassér}$

3 Output x_1, x_2, \dots, x_n

Detaljforklaring finner sted muntlig på forelesningen.

- La oss nå prøve å analysere kompleksiteten til denne algoritmen.
- Vi tar for oss ett og ett element fra den opprinnelige listen, og plasserer det på sin rette plass i forhold til den sorterte versjonen av den delen som kom foran.
- Det gir en hovedrunde med lengde n
- I hvert skritt i denne hovedrunden, må vi sammenlikne det objekter vi skal plassere med elementene i den ferdigsorterte delen av listen.
- Vi kan risikere å måtte sammenlikne det nye objektet med alle de som kom først.
- Hvis den opprinnelige listen kom ordnet helt motsatt av hva vi ønsker, skjer dette hver gang.

- Det vil gi oss

$$1 + 2 + \dots + (n - 1) = \frac{(n - 1)n}{2}$$

antall sammenlikninger.

- Siden det er disse sammenlikningene som er mest tidkrevende, kan vi konkludere med at tidskompleksiteten til denne algoritmen er $O(n^2)$.
- Er det mulig å være mer effektiv?
- Når vi skal sortere en liste med n elementer, er vi nødt til, på en eller annen måte å plassere alle n elementer på riktig plass.
- Det sier seg selv at dette må skje i omtrent n omganger.
- I den algoritmen vi så på brukte vi i gjennomsnitt $\frac{n}{2}$ antall sammenlikninger for å plassere et objekt i en allerede ordnet liste, i det verste tilfellet.
- Her er det rom for betydlige forbedringer.
- La oss se på et eksempel.
- Vi har gitt en ordnet liste på 16 objekter, eksempelvis tallene

1, 3, 7, 9, 12, 14, 22, 23, 25, 31, 37, 40, 41, 44, 47, 50

og vi vil finne plassen til tallet 8 i denne listen på en måte som kan inngå i en effektiv algoritme.

- Hvis vi bruker metoden fra i sted, vil vi foreta 14 tester.
- Etter den nye metoden vil vi starte med å sette det nye tallet inn i midten:
1, 3, 7, 9, 12, 14, 22, 23, 8, 25, 31, 37, 40, 41, 44, 47, 50
- Vi ser at midten er for langt oppe, så vi hopper ned til midten av den delen av listen som ligger under:
1, 3, 7, 9, 8, 12, 14, 22, 23, 25, 31, 37, 40, 41, 44, 47, 50
- Tallet ligger fremdeles for høyt, så vi gjør det samme en gang til:
1, 3, 8, 7, 9, 12, 14, 22, 23, 25, 31, 37, 40, 41, 44, 47, 50

- Nå kom vi for langt ned, så vi flytter oss opp igjen, halvparten så langt som vi flyttet sist.
- Det gir
1, 3, 7, 8, 9, 12, 14, 22, 23, 25, 31, 37, 40, 41, 44, 47, 50
- Ved systematisk å omtrent halvere den delen av den opprinnelige listen det nye objektet skal plasseres, vil antall trinn i plasseringsalgoritmen reduseres fra å være proporsjonal med n til å bli proporsjonal med antall sifre i n (spiller det noen rolle om vi snakker om binær representasjon eller dekadisk representasjon?)
- I boka står det en pseudokode for en sorteringsalgoritme basert på dette prinsippet.
- Det er ikke noe stort poeng å gjengi denne koden her så sent i semesteret.
- Poenget her at vi trenger en notasjon for å kunne snakke om tidskompleksiteten til algoritmen.

Definisjon.

Hvis n er et tall, lar vi

$$\lg(n)$$

være tallet m slik at $2^m = n$.

Vi kan kalle dette for binærlogaritmen til n .

- For alle praktiske formål i kompleksitetsteori, kunne vi brukt funksjonen som gir antall sifre i binærrepresentasjonen av n i stedet for.
- Den mer effektive sorteringsalgoritmen vil ha en tidskompleksitet som er $O(n \cdot \lg(n))$, n fordi vi fortsatt må plassere n objekter på riktig plass, men $\lg(n)$ fordi dette er tidskompleksiteten til den nye plasseringsalgoritmen.

Gjennomførbare algoritmer

- Vi har lovet at vi skal lære å vurdere om en algoritme kan gjennomføres i løpet av realistisk tid.
- Som de gode matematikere vi har blitt skal vi selvfølgelig gi en presis definisjon av hva som menes med en gjennomførbare eller overkommelig algoritme.
- Vi har sett på algoritmer hvor kompleksiteten er $O(n \cdot \lg(n))$, $O(n^{\frac{3}{2}})$ og $O(n^2)$.
- Alle disse er gjennomførbare.
- Vi skal se på noen algoritmer som ikke er gjennomførbare for store input.

Eksempel.

- Vi har laget en algoritme som avgjør om et utsagnslogisk uttrykk er en tautologi eller ikke.

- Den består i at vi skriver opp sannhetsverditabellen til uttrykket.
- Hvis n er antall symboler i uttrykket, vil antall søyler i tabellen i verste fall være $O(n)$, mens antall linjer i verste fall er $O(2^n)$.
- Tidskompleksiteten av sannhetsverditabellmetoden er altså i $O(n \cdot 2^n)$, og for store input er dette ikke gjennomførbart.

Eksempel.

- Det finnes ingen virkelig effektiv metode for å avgjøre om et naturlig tall er et primtall på, og de som er lette å forstå er i alle fall ikke effektive.
- Siden det er størrelsen av input som teller, og det er antall bits i binærrepresentasjonen av tallet, er det antall sifre i input som er utgangspunktet for å vurdere kompleksiteten.
- Den naive måten å undersøke om n er et primtall på er å undersøke om n har noen faktor m med $2 \leq m \leq \sqrt{n}$.

Eksempel (Fortsatt).

- Det holder selvfølgelig å gjøre dette for primtallene mellom 2 og \sqrt{n} , men da må vi kaste bort tid på å bestemme hvilke av disse tallene som er primtall, så det er ikke nødvendigvis så lurt.
- Hvis k er antall sifre i n , er $\frac{k}{2}$ omtrent antall sifre i \sqrt{n} , og det er omtrent $n^{\frac{k}{2}}$ antall divisjoner vi må utføre for å bestemme om n er et primtall eller ikke.
- I kryptografi er vi interesserte i primtall med hundre sifre eller mer, eller helst i produkter av to eller tre slike primtall.
- Da vil de naive metodene sprengte alle grenser for anstendig kompleksitet.

Eksempel.

- La G være en sammenhengende graf.
- Hvordan skal vi gå frem for å bestemme om grafen har en Hamiltonsti, det vil si en sti som er innom hver node nøyaktig en gang?
- Hvis n er antall noder i grafen, vil en Hamiltonsti ha $n - 1$ kanter
- Det finnes ingen kjent måte å undersøke om G har en Hamilton-sti på som er vesentlig mer effektive enn den naive, prøv alle stier med $n - 1$ kanter og se om en av dem tilfeldigvis skulle være en Hamiltonsti.

Eksempel (Fortsatt).

- I det verste tilfellet er antall stier i G med $n - 1$ kanter $O\left(\binom{n^2}{n-1}\right)$, det vil si

$$\frac{(n^2)!}{(n^2 - n + 1)!(n - 1)!}$$

- Dette er et tall som faktisk er større enn 2^{n-1} , så algoritmen er ikke imponerende effektiv.

Definisjon.

Vi sier at en algoritme er gjennomførbar (tractable på engelsk) hvis tidskompleksiteten er $O(n^k)$ for en k .

Merk.

- Det er flere grunner til at man har falt ned på dette som en fornuftig definisjon.
- Tidligere erfaringer tilsa at hvis en algoritme er gjennomførbar i henhold til denne definisjonen, kan den brukes i praksis. k ligger gjerne rundt tre eller lavere.
- Ganske overraskende viste en gruppe indere for noen år siden at det finnes en algoritme som avgjør om et tall er et primtall eller ikke som faller inn under denne definisjonen, men der var k (og konstanten c) så stor at algoritmen hadde mer teoretisk enn praktisk verdi.
- Definisjonen er også ganske robust, selv om forskjellige matematiske modeller for hva en beregning består i kan gi forskjellige verdier på graden.

- Vi skal avslutte disse forelesningene med å snakke bittelitegrann om **P** og **NP**.
- **P** er klassen av problemer som kan løses i polynomisk tid, det vil si de som kan løses av en gjennomførbar algoritme slik vi har definert det.
- Eksempler på problemer som ligger i **P** er om en graf er sammenhengende og om den har en Eulerkrets, om to termer lar seg unifisere, om et uttrykk svarer til en term på polsk form og etterhvert om et tall er et primtall eller ikke (det kom som en overraskelse).
- **NP** er grovt sagt klassen av problemer hvor vi med flaks bare trenger å bruke polynomisk tid for å løse det den ene veien, mens vi tilsynelatende bruker eksponensiell tid om løsningen går den andre veien.
- Hvis G er en graf, og noen streker opp en Hamiltonsti, er det raskt å få bekreftet at det er en Hamiltonsti det er, mens hvis det ikke finnes noen Hamiltonsti trenger vi lang tid.

- Hvis A er et uttrykk som ikke er en tautologi, kan vi få vite det veldig fort hvis vi tilfeldigvis prøver den fordelingen av sannhetsverdier som gjør utsagnet usant, mens vi fortsatt må skrive ut hele sannhetsverditabellen hvis utsagnet er en tautologi.
- Det store åpne problemet er om disse mengdene av problemer er de samme, eller om det finnes problemer som er i **NP** men ikke i **P**.
- Dette er et av de syv milleniumsproblemene i matematikk, og det er en dusør på $\$10^6$ for hvert av de seks som står fortsatt uløst.

- Det finnes mange temaer som faller inn under sekkebetegnelsen diskret matematikk som vi ikke har tatt opp ennå.
- Det finnes også temaer det ville vært naturlig for oss å ta opp, eksempelvis sammenhengen mellom trerekursjon og kompleksitet.
- Siden vi dog trenger noe tid til repetering, og pensum har blitt passe stort, gjenstår det bare å si at presentasjonen av nytt stoff i MAT1030, våren 2008, har tatt

SLUTT

Forelesning 32

Repetisjon

Dag Normann - 21. mai 2008

Streik?

- Det er muligheter for streik i offentlig sektor fra midnatt, natt til fredag.
- I tilfelle streik, vil enkelte ansatte ved UiO bli tatt ut.
- I første omgang har det som konsekvens at Vaktentralen stenger.
- Det betyr igjen at alle heiser står. De vil stanse torsdag kl. 13.00 om forhandlingene ikke er avsluttet til da.
- Det foreligger ingen signaler om at eksamen i MAT1030 vil bli berørt av en eventuell streik.

Repetisjon

Mandag fortalte jeg hvilke forelesninger det er som inneholder eksamensrelevant stoff som ikke står i læreboka.

Det skader ikke å gjenta det.

Det er

- Forelesning 17, 10. mars.
 - Forelesning 18, 12. mars.
 - Forelesning 26, 28. april.
 - Forelesning 27, 30. april.
 - Forelesning 28, 5. mai.
-
- I denne forelesningen skal vi systematisk gå igjennom alle delene av pensum, og vi skal trekke frem de ferdighetene man må beherske for å være sikker på å greie 90% av eksamenssettet.
 - For å være sikker på greie 100% må man kunne alt som er oppgitt som pensum.
 - Oppgaver kan hentes fra hele pensum, selv om hovedvekten legges på stoff som ikke er testet gjennom de to obligatoriske oppgavene.
 - Hvis man kan løse alle oppgaver av den typen som er gitt som treningsoppgaver gjennom semesteret, ligger man meget godt an.
 - Vi kommer ikke med vilje til å gi oppgaver som har vært gitt før.
-
- Denne gjennomgangen vil ta to timer, eller mer.
 - Vi skal se på noen eksempler, men det er ikke slik at alle eksemplene dekker oppgaver som blir gitt til eksamen, og heller ikke slik at vi illustrerer alle eksamensoppgavene med eksempler her.

- Hvis det er stoff man vil ha gjennomgått mer grundig, kan vi gjøre det neste uke.
- Jeg stiller opp mandag og onsdag for å finne ut av hvem som vil høre om hva.
- Det er også mulig å sende meg e-post om aktuelle temaer for neste uke.
- Uansett vil vi da legge vekt på eksempler, helst som tavleregning.

Kapittel 1

- I Kapittel 1 så vi på algoritmebegrepet generelt og kontrollstrukturer spesielt.
- Vi har brukt kontrollstrukturer når det har passet seg slik gjennom hele semesteret.
- Viktige deler av læringsmålet i dette emnet er at studentene skal
 - * Kunne utarbeide en kontrollstruktur som en nærmere presisering av en algoritme.
 - * Kunne lese en kontrollstruktur og forstå sammenhengen mellom inputverdiene og outputverdiene.
 - * I tilknytning til Kapittel 13, kunne vurdere tidskompleksiteten.
- En typisk mulig oppgave om dette ble gitt i ekstrasettet til Uke 21.

Kapittel 2 og 3

- Kapittel 2 og 3 omhandler tallsystemer og prinsipper for digital representasjon av hele tall og reelle tall.
- Mye av dette stoffet er behandlet mer grundig i andre emner.
- Forståelsen av digital representasjon ble forsiktigvis testet i det første obligatoriske oppgavesettet.
- Vi har etterhvert hatt bruk for å vise hvordan relasjoner, grafer, trær og andre dataobjekter kan representeres i en datamaskin.
- Det er lite aktuelt med en oppgave som behandler stoff fra disse kapitlene direkte, men man kan ha fordel av å forstå hvordan grafer, trær og andre objekter representeres, især i forbindelse med en eventuell oppgave i kompleksitetsteori.

Kapittel 4

- I dette kapitlet om logikk har vi konsentrert oss om
 - utsagnslogikk
 - kvantorer
 - føring av argumenter
- I tillegg er det en del snakk om betydningen av logikk i programmering.
- Dette siste er mest motiverende, men en motivasjon man bør ta med seg til senere emner.
- I utsagnslogikk har vi lagt mest vekt på det formelle språket, hvor vi tar utgangspunkt i noen utsagnsvariable, og bygger opp utsagn fra utsagnsvariablene ved å bruke bindeordene (eller konnektivene) \neg , \vee , \wedge , \rightarrow og \leftrightarrow .

- Et typisk eksempel kan være

$$A = \neg(p \rightarrow q) \rightarrow \neg q \vee p.$$

- Vi lærte hvordan vi kunne spare på parenteser, slik at vi vet hvordan parentesene egentlig skal settes i eksemplet over.
- Vi skal kunne avgjøre om et utsagn som det over er en tautologi, en kontradiksjon eller ingen av delene.
- Den sikreste måten å gjøre det på er å sette opp en sannhetsverditabell.
- Tabellen til utsagnet over finner vi på neste side.

p	q	$p \rightarrow q$	$\neg(p \rightarrow q)$	$\neg q$	$\neg q \vee p$	A
T	T	T	F	F	T	T
T	F	F	T	T	T	T
F	T	T	F	F	F	T
F	F	T	F	T	T	T

Vi ser at utsagnet er en tautologi.

- Vi lærte også om bruk av kvantorer.
- Kvantorene \exists - "det eksisterer", og \forall - "for alle", brukes for å gi matematiske definisjoner og setninger en presis formulering.
- Kvantorer ble ikke noe sentralt tema dette året, men vi fikk bruk for dem da vi skulle undersøke om f er $O(g)$ for bestemte funksjoner f og g .
- Det er ofte viktig å ha preise definisjoner når man skal vise at en egenskap ikke holder.
- Det siste temaet vi tok opp under dette kapitlet var bevisformer hvor vi skiller mellom direkte bevis og indirekte eller kontrapositive bevis.
- I et kontrapositivt bevis antar vi at den påstanden vi vil vise er usann, og så beviser vi at noen av forutsetningene heller ikke kan være sanne.
- Vi har sett på noen eksempler på bevis, men har ikke laget noe stort nummer av det.
- Det er meningen at dere skal kunne gjennomføre et enkelt resonement om dere blir bedt om det.

Kapittel 5

- Kapittel 5 tar opp et bredt tema over få sider.
- Først har vi en innføring i Boolsk mengdelære med snitt, union, komplement og mengdedifferens.
- Det er viktig at dere kjenner definisjonene av $A \cap B$, $A \cup B$, \bar{A} og $A - B$.
- To mengder er like hvis de har de samme elementene.
- $A \subseteq B$, A er inneholdt i B , hvis alle elementene i A også er elementer i B .
- Når vi snakker om komplementet \bar{A} , har vi alltid antatt at vi har en universell mengde \mathcal{E} .

- Hvis vi har et Boolsk uttrykk hvor det inngår to eller tre vilkårlige mengder, kan vi undersøke egenskapene ved dette uttrykket ved å bruke et Venndiagram
- Et eksempel på en mulig oppgave kan være å bruke et Venndiagram til å avgjøre om følgende inklusjon alltid vil holde:

$$(A \cap \bar{B}) - C \subseteq \overline{C - (B \cup A)}.$$

- Vi tar den på tavla.
- Kardinaliteten til en mengde er et finere ord for hvor mange elementer mengden har.
- Vi har lært om ordnede par, kartesisk produkt og om potensmengder.
- Potensmengden til A er mengden av alle delmengder av A .
- Dere bør vite at hvis A har n elementer, så har A 2^n delmengder, og A^2 har n^2 elementer.
- Siste delen av dette kapitlet omhandler relasjoner.
- En relasjon på en mengde A er en delmengde R av A^2 .
- Vi har sett på fem egenskaper en relasjon kan ha
 - Transitiv: $xRy \wedge yRz \Rightarrow xRz$ for alle x, y og z .
 - Refleksiv: xRx for alle x .
 - Irrefleksiv: Ikke xRx for noen x .
 - Symmetrisk: $xRy \Rightarrow yRx$ for alle x og y .
 - Antisymmetrisk: $xRy \wedge yRx \Rightarrow x = y$ for alle x og y .
- I enkle tilfeller bør vi kunne bestemme om en relasjon er refleksiv, om den er symmetrisk og om den er transitiv.
- En relasjon som er refleksiv, symmetrisk og transitiv kalles en ekvivalensrelasjon.
- En ekvivalensrelasjon R på en mengde A vil dele A opp i disjunkte ekvivalensklasser av parvis ekvivalente objekter.
- I enkle situasjoner bør dere være i stand til å beskrive ekvivalensklassene til en ekvivalensrelasjon.

Eksempel.

La $A = \{2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 27, 32, 36, 48\}$.

Hvis n og m er i A , lar vi nRm hvis n og m har like mange faktorer når de er fullstendig faktoriseret.

Kan vi finne ekvivalensklassene?

Ekvivalensklassene vil svare til delmengder av A som har den interessante egenskapen felles.

Den interessante egenskapen er her antall faktorer.

Eksempel (Fortsatt).

- 1 En faktor: $\{2, 3\}$
- 2 To faktorer: $\{2^2, 2 \cdot 3, 3^2\} = \{4, 6, 9\}$
- 3 Tre faktorer: $\{2^3, 2^2 \cdot 3, 2 \cdot 3^2, 3^3\} = \{8, 12, 18, 27\}$
- 4 Fire faktorer: $\{2^4, 2^3 \cdot 3, 2^2 \cdot 3^2\} = \{16, 24, 36\}$
- 5 Fem faktorer: $\{2^5, 2^4 \cdot 3\} = \{32, 48\}$

Kapittel 6

- Selv om funksjonsbegrepet gjennomsyrrer mye av matematikken, har vi ikke lagt mye vekt på generell funksjonslære.
- Funksjonsbegrepet gir oss et språk som vi kan bruke i andre deler av matematikken og informatikken.
- Ved hjelp av det språket kan vi gi mening til at to grafer "egentlig" er like, det vil si isomorfe.
- En algoritme vil gjerne definere en funksjon, funksjonen som fra en input-verdi gir oss den tilsvarende outputverdien.
- Vi brukte også funksjoner for å kunne måle tidskompleksiteten til en algoritme.
- Til eksamen kan det være aktuelt å kjenne til hvordan man setter sammen to funksjoner, kjenne igjen en injektiv (eller surjektiv) funksjon om man får bruk for det, men læringsmålet er i hovedsak å kunne bruke funksjonsbegrepet i sammenhenger utenom Kapittel 6, så det blir ikke lagt mye vekt på rendyrkede oppgaver om vilkårlige funksjoner.

Kapittel 7

- Et av de viktigste kapitlene i pensum er kapitlet om induksjon og rekursjon.
- Vi starter med å se på hvordan man definerer en følge ved rekursjon:
$$f(1) = 1$$
$$f(n) = 2f(n-1) + 1 \text{ når } n > 1.$$
- Poenget med rekursjon er at ved hjelp av disse to linjene har vi bestemt hva $f(n)$ må være for alle $n \geq 1$.
- Vi har at $f(2) = 2f(1) + 1 = 2 + 1 = 3$
- Vi har at $f(3) = 2f(2) + 1 = 6 + 1 = 7$
- Vi har at $f(4) = 2f(3) + 1 = 14 + 1 = 15$
- Slik kan vi fortsette.
- Hvis vi nå ser at $f(1) = 2^1 - 1$, at $f(2) = 2^2 - 1$, at $f(3) = 2^3 - 1$ og at $f(4) = 2^4 - 1$, er det nærliggende å tro at $f(n) = 2^n - 1$ for alle $n \in \mathbb{N}$.

- Det er til og med nærliggende å tro at denne egenskapen har noe med hvordan vi definerte f å gjøre.
- Vi bruker et induksjonsbevis for å vise at $f(n) = 2^n - 1$ for alle n .
- Induksjonsbevis er delt opp i induksjonstart og induksjonskritt
- Induksjonstarten tilsier at vi skal vise at $f(1) = 2^1 - 1$, og det holder siden både høyre og venstre sider har verdi 1.
- I induksjonskrittet antar vi at $f(n - 1) = 2^{n-1} - 1$, hvor $n > 1$, og vi skal vise at da er $f(n) = 2^n - 1$.
- I dette tilfellet er det rett frem.
- En ulempe ved dette formatet på induksjonsbevis er at det gjør det lett å bli opptatt av formen bevisene får på bekostning av forståelsen av hvorfor bevisformen gir riktige svar.
- Vi har derfor lagt vekt på å bruke rekursjon og induksjon i en mer generell sammenheng.
- Det ene tilfellet er hvor grunnlaget for induksjonen er mer enn et tall.
- Dette er systematisk gjennomført for rekurrenslikninger.
- Rekurrenslikninger er et takknemlig oppgavestoff, og vi skal ta et eksempel.

Eksempel.

- Anta at vi har gitt rekurrenslikningen

$$F(n) = F(n - 1) + 2F(n - 2).$$

- Denne likningen kan også skrives som

$$F(n) - F(n - 1) - 2F(n - 2) = 0.$$

- Vi finner den generelle løsningen av en slik likning ved å se på den karakteristiske likningen

$$r^2 - r - 2 = 0$$

som har løsninger $r = 2$ og $r = -1$.

Eksempel (Fortsatt).

- Den generelle løsningen av likningen er da

$$f(n) = A \cdot 2^n + B \cdot (-1)^n.$$

- Hvis vi nå i tillegg får vite at vi skal ha at $f(1) = 1$ og $f(2) = 11$, har vi fått to initialbetingelser
- Den rekursive definisjonen forteller oss at

$$F(3) = F(2) + 2F(1) = 11 + 2 = 13$$

$$F(4) = F(3) + 2F(2) = 13 + 22 = 35$$

OSV.

- Vi kan finne hele løsningen ved å løse likningene

$$A \cdot 2^1 + B \cdot (-1)^1 = 1$$

$$A \cdot 2^2 + B \cdot (-1)^2 = 11$$

som gir $A = 2$ og $B = 3$.

Eksempel (Fortsatt).

- Den spesielle løsningen er derfor

$$f(n) = 2 \cdot 2^n + 3 \cdot (-1)^n.$$

- Hvis vi nå blir bedt om å bekrefte denne formelen ved et induksjonsbevis, kan vi argumentere som følger:
- Ved at vi har løst de to førstegradslikningene, vet vi at formelen stemmer for $n = 1$ og for $n = 2$
- Vi lar nå $n > 2$ og vi antar at formelen stemmer for $n - 1$ og $n - 2$.
- Da kan vi bruke rekurrensdefinisjonen, regne litt, og se at formelen stemmer for n også
- Da kan vi konkludere med at vi har et induksjonsbevis.

- Vi så også på induktivt definerte strukturer generelt.
- Vi har en induktivt definert struktur hvis vi strukturen (eller mengden) er konstruert ved at vi har noen basisobjekter, og noen prinsipper for hvordan vi konstruerer nye objekter fra eksisterende objekter.
- Vi har sett på mengden av ord som bygget opp fra det tomme ordet ved etter tur å legge nye bokstaver til høyre for ordet.
- Vi har sett på mengden av utsagnslogiske uttrykk som bygget opp fra utsagnsvariable ved å bruke de logiske bindeordene.
- Etterhvert så vi på de binære trærne som bygget opp fra den enkle bladnoden ved hjelp av binære forgreninger.
- Hver gang vi har en induktivt definert struktur, har vi et grunnlag for å konstruere funksjoner ved rekursjon, og et grunnlag for å bevise setninger ved induksjon.
- Vi får da en rekursjon(induksjon)start for hvert basisobjekt og et rekursjon(induksjon)skritt for hver måte å konstruere nye objekter på.
- Vi har sett dette eksemplifisert gjennom konstruksjon av svak normalform og gjennom eliminering av \rightarrow og \leftrightarrow fra utsagnslogiske uttrykk, og vi så på eksempler som sammenbinding av ord, spelling av ord etc.
- Vi la stor vekt på bruk av trerekursjon i et senere kapittel.

Kapittel 9

- Avsnittet om kombinatorikk er lite sammenliknet med tilsvarende avsnitt i andre kurs i diskret matematikk.
- Mye av det som står her er dekket av pensum i Videregående skole.
- Man bør kunne kjenne binomialkoeffisientene, deres definisjon og bruksområde.
- Man bør også kjenne til formelen for hvor mange måter man kan velge ut k elementer i rekkefølge fra en mengde med n elementer på, men her legger vi ikke vekt på at man husker notasjonen.
- Pensum er nøyaktig det som står i boka, og vi bruker ikke mer tid på det her.

Kapittel 10

- I grafteori har vi lagt mest vekt på ordinære grafer, og mindre vekt på rettede grafer.
- Det er noen begreper det vil kunne være en fordel å kjenne til, som
 - Node
 - Kant
 - Graden til en node
 - Løkke
 - Parallell kanter
 - Enkel graf
 - Krets
 - Sti
 - Sammenhengende graf
- Vi har behov for å vite hvordan grafer representeres digitalt i Kapittel 13.
- Ellers er de viktigste ferdighetene fra dette kapitlet å kunne vurdere om en graf har en Eulersti eller en Eulerkrets.
- I de tilfellene hvor grafen har en Eulersti eller en Eulerkrets, skal man kunne finne en slik en.
- To grafer G og H er isomorfe hvis det finnes en bijeksjon f fra nodene til G til nodene til H og en bijeksjon g fra kantene til G til kantene til H slik at s er en kant mellom a og b hvis og bare hvis $g(s)$ er en kant mellom $f(a)$ og $f(b)$.
- Det er meningen at dere skal resonnerer rundt isomorfi mellom grafer.

Forelesning 33

Repetisjon

Dag Normann - 26. mai 2008

Innledning

- Onsdag 21/5 gjorde vi oss ferdige med det meste av den systematiske repetisjonen av MAT1030.
- Det som gjensto var kapitlene 11 om trær og 13 om kompleksitetsteori.
- I mellomtiden er jeg bedt om å repetere stoffet om bevisstrer ekstra grundig.
- Det vil jeg forsøksvis gjøre som en del av denne foil-baserte repetisjonen.
- Etter at disse foilene er gjennomgått vil eventuell repetisjon foregå som tavleundervisning.

Kapittel 11

- En sykel i en graf er en sti som begynner og slutter i samme node, men som ellers ikke er innom samme node to ganger.
- En sykel kan da heller ikke inneholde samme kant to steder.
- Et tre er en sammenhengende graf som ikke inneholder noen sykel.
- Et tre vil alltid være en enkel graf, ettersom to parallelle kanter danner en sykel.
- I et tre er alltid antall kanter en mindre enn antall noder.
- Dette er en viktig kunnskap for å kunne besvare enkle spørsmål.

- Vår første anvendelse av trær er i forbindelse med enkle, vektete grafer.
- En graf er vektet hvis hver kant er utstyrt med et ikke-negativt tall, en vekt.
- Et utspennende tre i en enkel graf er en delgraf som omfatter alle nodene og som er et tre.
- Prims algoritme står sentralt i pensum, og vil normalt bli tema for en eksamensoppgave.
- Med Prims algoritme skal man finne et utspennende tre i en vektet graf som har minimal samlet vekt.
- I Prims algoritme starter man i et vilkårlig punkt.
- Deretter bygger man opp et tre, ved i hvert skritt å legge en kant som forbinder den delen av treet man har bygget opp med en *ny* node.
- (I boka er dette formulert som at man ikke innfører noen sykel.)

- Den andre algoritmen vi har sett på i forbindelse med vektete grafer er Dijkstras algoritme.
- Dijkstras algoritme er også eksamensrelevant.

- Poenget her er å starte med en utvalgt node, og så finne det utspennende treet som gir minimal avstand fra hver av de andre nodene til den utvalgte.
- Siden to noder i et tre er forbundet med en og bare en sti, lar vi "avstand" mellom to noder bety summen av vektene på kantene i denne stien.
- Dijkstras algoritme lar oss også bygge opp treet node for node og kant for kant, men i hvert skritt legger vi nå til en kant til en ny node som gir oss en minimal ny avstand til sentrumsnoden.
- De andre typene trær vi har sett på er trær med rot, merkede trær og binære trær.
- Vi har spesielt lagt vekt på å studere syntakstrær og bevistrær.
- I begge disse tilfellene snakker vi om merkede, binære trær, og dermed spesielt om trær med rot.
- Et bevistre gir oss en mulighet for å analysere om et utsagn på svak normalform er en tautologi eller ikke.
- Hver node er merket med en disjunksjon (\vee) av formler på svak normalform, hvor hvert ledd i disjunksjonen enten er en literal, det vil si en utsagnsvariabel eller negasjonen av en utsagnsvariabel, eller en konjunksjon (\wedge).
- En bladnode hvor det forekommer både en utsagnsvariabel og negasjonen dens, kalles et aksiom.

- Vi minner først om hva det vil si at et utsagn er på svak normalform.
- I følge definisjonen er et utsagn på svak normalform hvis vi bare bruker bindeordene \neg , \vee og \wedge , og hvor \neg alltid forekommer rett foran en utsagnsvariabel.
- Eksempelvis er

$$(\neg p \vee q) \wedge (p \vee \neg q)$$

på svak normalform, mens

$$\neg(p \vee q)$$

ikke er det.

- Vi kan se på mengden av uttrykk på svak normalform som en induktivt definert mengde, ved at basisuttrykkene er literaler som p og $\neg q$, og hvor vi bare bruker \wedge og \vee for å bygge opp mer komplekse uttrykk.
- Bevistrærne tar utgangspunkt i at utsagn på formen $A_1 \vee \dots \vee A_n$ vil være en tautologi hvis det finnes en i og j slik at $A_i = \neg A_j$.
- Vi opphøyer derfor slike utsagn til aksiomer. De er både tautologier, og det er effektivt å la en datamaskin teste om et utsagn på svak normalform oppfyller denne egenskapen.
- La nå $D \vee A \vee C$ og $D \vee B \vee C$ være to utsagn, og anta at vi vet at disse to utsagnene er sanne i en gitt situasjon.
- Da vet vi at $D \vee (A \wedge B) \vee C$ også er sann.
- Dette kan vi se ved å se på en sannhetsverditabell med 8 linjer!
- Vi skriver ut tabellen på tavla.
- I alle linjene hvor både $D \vee A \vee C$ og $D \vee B \vee C$ får verdien T, vil også $D \vee (A \wedge B) \vee C$ få verdien T.

- Da kan vi opphøye
 $D \vee A \vee C$ og $D \vee B \vee C$ kan vi slutte $D \vee (A \wedge B) \vee C$
 til en logisk regel (hvor det ikke trenger å stå noe på plassen til D eller til C).
- Et bevistre er et merket, binært tre, hvor alle nodene er merket med utsagn på svak normalform, hvor alle bladnodene er aksiomer, og hvor merket til en forgreningsnode alltid følger fra merkene til barna ved regelen over.
- Det er to fordeler med slike bevistrær:
 1. Det finnes effektive algoritmer for å teste om et merket tre er et bevistre eller ikke.
 2. Hvis A er et utsagn på svak normalform, finnes det en lett forståelig strategi for å lage et bevistre for A (det vil si at rotnoden er merket med A) hvor vi ofte raskt kan bestemme om A er en tautologi eller ikke.
- I de verste tilfellene trenger vi fortsatt eksponensielt lang tid, men de verste tilfellene oppstår ofte ikke i praktiske anvendelser.
- Det vi forsøksvis har prøvd å lære bort i disse forelesningene, etter ønske fra grupperinger ved Ifl, er hvordan vi lager et bevistre fra et utsagn.
- Vi skal se på et par eksempler til.

Eksempel.

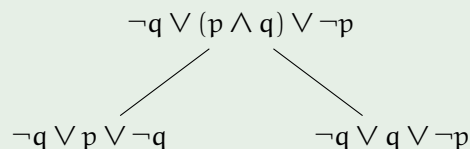
- $$\neg q \vee (p \wedge q) \vee \neg p$$
- Hvis vi prøver å bruke den logiske regelen baklengs, se vi at dette utsagnet er en konsekvens av utsagnene

$$\neg q \vee p \vee \neg p$$
- og

$$\neg q \vee q \vee \neg p.$$

Eksempel (Fortsatt).

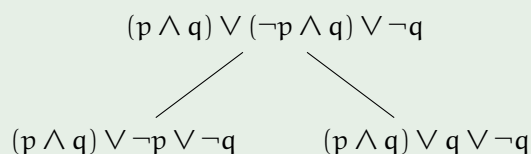
- Da kan vi starte konstruksjonen av et bevistre ved å bruke tilsvarende forgrening:



- Vi ser at vi allerede har fått aksiomer på de to barna, så vi kan bruke dem som bladnoder, og har et bevistre.

Eksempel.

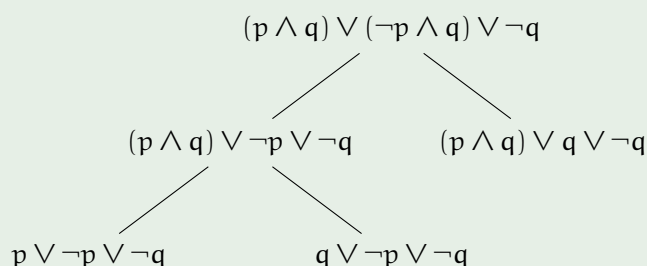
- $(p \wedge q) \vee (\neg p \wedge q) \vee \neg q$
- Her er det to ledd med \wedge , og vi kan velge hvilket vi vil løse opp først.
- Vi velger det andre og får forgreningen



- Her er ikke venstre barn et aksiom, men høyre barn er det.

Eksempel (Fortsatt).

- Vi prøver oss derfor med en ytterligere forgrening fra venstre barn.
- Det gir treet



- Vi ser at dette treet er et bevistre.

- Denne metoden vil alltid gi oss et tre.
- Hvis vi ender opp med en bladnode som ikke er et aksiom, er utsagnet vi startet med ikke en tautologi.
- Hvis vi ender opp med aksiomer i alle bladnodene, er utsagnet en tautologi.
- Ytterligere utdypinger kan vi ta på tavlen om ønskelig.
- Vi har ikke fått tid til å trene så mye på arbeid med bevistrær som ønskelig, men de inngår i pensum likevel.
- Hvis vi utvider språket til å omfatte kvantorer, spiller slike bevistrær en stor rolle, både i de teoretiske studiene og i praktisk bevissøk.
- I forbindelse med syntakstrærne har vi sett på infiks notasjon, polsk notasjon og baklengs polsk notasjon.
- Det er meningen at man skal kunne skrive ned syntakstreet til en term eller et uttrykk, og ved hjelp av det kunne finne frem til hvordan termen eller uttrykket ser ut med hhv. infiks, polsk og omvendt polsk notasjon.

- I denne forbindelse har vi også sett på unifisering og unifiseringsalgoritmen.
- Unifiseringsproblemet generelt går ut på at vi har n par t_1, s_1 opp til t_n, s_n av termer i et språk.
- I disse termene kan det forekomme variable x_1, \dots, x_k .
- Er det mulig å erstatte alle variablene x_1, \dots, x_k med termer r_1, \dots, r_k slik at begge sider av hvert enkelt par blir syntaktisk like?
- Unifisering er eksamensrelevant.
- Hvis det blir gitt en oppgave om unifisering til eksamen, av den type vi er vant med, vil vi etter oppfordring fra en student bruke $*$ for multiplikasjon i stedet for \times , for lettere å kunne skille multiplikasjon fra variabelen x .

Kapittel 13

- Det siste kapitlet omhandler algoritmer og kompleksitet.
- Gjennomgangen av dette kapitlet gikk litt fort på slutten, men det kan være aktuelt med en enkel oppgave fra dette stoffet.
- Det som da vil være aktuelt er å finne frem til tidskompleksiteten til algoritmen bak en enkel pseudokode, å kunne beskrive denne ved hjelp av O -notasjonen, og å kunne vurdere om algoritmen er gjennomførbar (tractable) eller ikke.
- Husk at en algoritme er, per definisjon, gjennomførbar hvis tidskompleksiteten er $O(n^k)$ for et naturlig tall k .
- Vi har tidligere sagt at det ikke er aktuelt med eksamensoppgaver som i vanskelighetsgrad og form går ut over det som er gitt som øvingsoppgaver gjennom semesteret.
- For Kapittel 13 sin del er dette fire oppgaver fra boka pluss en ekstraoppgave lagt ut på nettet.
- I går fant jeg en oppfordring om å komme med et par eksempler til.
- Disse følger nå.

Oppgave.

Betrakt følgende pseudokode:

```

1 Input  $k$  [ $k \in \mathbb{N}$ ]
2 Input  $n_1, \dots, n_k$  [Seksens av hele tall med standard digital representasjon]
3  $x \leftarrow n_1$ 
4 For  $i = 2$  to  $k$  do
    4.1 If  $x < n_i$  then
        4.1.1  $x \leftarrow n_i$ 
5 Output  $x$ 

```

Oppgave (Fortsatt).

- a) Forklar sammenhengen mellom input og output.
- b) Finn et uttrykk for tidskompleksiteten.

Løsning

- a) Siden vi hele veien lar x ha verdien til den største n_i lest så langt, vil x til slutt bli det største av tallene n_1, \dots, n_k .
- b) Siden vi bruker standard digital representasjon på tallene n_i , har antall bits vi bruker til å representere hvert enkelt tall en fast lengde.

Derfor er k et mål på hvor stort input er.

Den mest tidkrevende enkeltoperasjonen er sammenlikningen av x og n_i , en operasjon vi utfører k ganger.

Tidskompleksiteten blir da $O(k)$.

Oppgave.

Betrakt følgende pseudokode:

```
1 Input  $k$  [ $k \in \mathbb{N}$ ]  
2 Input  $n$  [ $n \in \mathbb{N}$ ]  
3  $x \leftarrow n$   
4 For  $i = 2$  to  $k$  do  
    4.1 If  $x$  like tall then  
        4.1.1  $x \leftarrow \frac{x}{2}$   
        else  
        4.1.2  $x \leftarrow 3x + 1$   
5 Output  $x$ 
```

Oppgave (Fortsatt).

Bestem tidskompleksiteten til algoritmen over.

Løsning

Når ikke annet er nevnt, skal vi anta at input er gitt på binær form.

La m være antall bits vi bruker til å representere input.

m er av størrelsesorden $\lg(k) + \lg(n)$.

De tre leddene i hovedløkka er omtrent like arbeidskrevende, så det blir lengden på hovedløkka som bestemmer tidskompleksiteten.

Lengden på denne løkka er $k - 1$, som er $O(2^m)$ hvor m er størrelsen på input.

Siden vi uansett om prosessen stabiliserer seg eller ikke insisterer på å gjennomføre $k - 1$ runder, vil $O(2^m)$ være tidskompleksiteten.

Plenumsregning 1

Kapittel 1

Roger Antonsen - 17. januar 2008

Velkommen til plenumsregning for MAT1030

- Torsdager 10:15–12:00
- Gjennomgang av ukeoppgaver
- Gjennomgang av eksempler fra boka
- Litt repetisjon fra forelesningen
- Spørsmål og svar
- *Forsøk på oppgavene selv først!*

Repetisjon: Algoritmer og pseudokode

- En algoritme er en oppskrift som forteller oss hvordan vi skritt for skritt skal kunne oppnå et resultat eller løse et problem.
- Det skal ikke kreves intelligens eller forståelse for å følge den.
- En pseudokode er en måte å beskrive en algoritme på.
- Hvert steg i algoritmen skal være beskrevet på en *entydig* måte.

Eksempel fra boka

Eksempel (1.1.1).

Skriv en algoritme som regner ut arealet av en sirkel, gitt radiusen.

Løsning

1. Input r [r er radiusen til sirkelen.]
2. $areal \leftarrow \pi r^2$
3. Output areal

- Hvert steg i algoritmen er nummerert.

- Kommentarer skrives mellom [].
- Symbolet \leftarrow betegner *tilordning*.

Repetisjon: Kontrollstrukturer

- En kontrollstruktur brukes for å styre hvordan, og hvorvidt, de enkle instruksjonene i en pseudokode skal utføres.

For-do

1. $sum \leftarrow 0$
2. **For** $i = 1$ **to** 6 **do**
 - 2.1. $sum \leftarrow sum + i$

Kjøring

Steg	sum	i
—	—	—
1	0	—
2	0	1
2.1	1	1
2	1	2
2.1	3	2
2	3	3
2.1	6	3
2	6	4
2.1	10	4
2	10	5
2.1	15	5
2	15	6
2.1	21	6
—	21	—

If-then

1. **If** $x < 0$ **then**
 - 1.1. $x \leftarrow -x$

If-then-else

1. **If** $x \geq 0$ **then**
 - 1.1. $y \leftarrow \sqrt{x}$
- else**
 - 1.2. Output ' \sqrt{x} fins ikke'

While-do

1. **While** $svar \neq 1$ **do**
 - 1.1. Input $svar$

Repeat-until

1. $i \leftarrow 0$
 2. **Repeat**
 - 2.1. $i \leftarrow i + 1$
 - 2.2. Input x_i
- until** $x_i = 0$

Eksempler fra boka

Eksempel (1.2.1/1.2.2).

Finn det minste tallet i en liste av tall.

Løsning

1. *Input the number of values* n
2. *Input the list of numbers* x_1, \dots, x_n
3. $\min \leftarrow x_1$
4. **For** $i = 2$ **to** n **do**
 - 4.1. **If** $x_i < \min$ **then**
 - 4.1.1. $\min \leftarrow x_i$
5. *Output* \min

Kjøring

Steg	min	i	n	x_1	x_2	x_3
—	—	—	—	—	—	—
1	—	—	3	—	—	—
2	—	—	3	5	4	8
3	5	—	3	5	4	8
4	5	2	3	5	4	8
4.1	5	2	3	5	4	8
4.1.1	4	2	3	5	4	8
4	4	3	3	5	4	8
4.1	4	3	3	5	4	8
5	4	—	3	5	4	8

Reserverte ord og utrykk i pseudokode

- If
- then
- else
- For
- to
- do
- While
- Repeat
- until
- and
- or
- not
- Input
- Output
- \leftarrow
- true
- false
- [] (eller {} i boka)
- vanlig matematiske uttrykk (f.eks. $x < 0$ og \sqrt{x})

Eksempler fra boka

Eksempel (1.2.3).

Lag en algoritme som sjekker om en streng $c_1c_2 \dots c_n$ av n tegn består av kun sifre, eller om ikke-sifre forekommer, og skriv ut en passende melding.

- Hvis input er 12345 skal algoritmen returnere meldingen: Strengen inneholder kun sifre.
- Hvis input er 123@45 skal algoritmen returnere meldingen: Strengen inneholder tegn som ikke er sifre.

Utkast 1

1. $i \leftarrow 1$; *ikkesiffer_oppdaget* \leftarrow false
 2. **Repeat**
 - 2.1. **If** c_i ikke er et siffer **then**
 - 2.1.1. *ikkesiffer_oppdaget* \leftarrow true
 - 2.2. $i \leftarrow i + 1$
- until** *ikkesiffer_oppdaget* = true

- Identifikatorer/variable skrives i *kursiv* uten mellomrom.
- *ikkesiffer_oppdaget* kalles gjerne for en *logisk* (eller Boolsk) variabel, siden den kun vil ta verdiene true eller false.

- Kunne ha skrevet '**until** *ikkessiffer_oppdaget*' i siste linje.
- Denne vil ikke *terminere* hvis strengen består av kun sifre.

Utkast 2

1. $i \leftarrow 1$; *ikkessiffer_oppdaget* \leftarrow false
 2. **Repeat**
 - 2.1. **If** c_i ikke er et siffer **then**
 - 2.1.1. *ikkessiffer_oppdaget* \leftarrow true
 - 2.2. $i \leftarrow i + 1$
- until** *ikkessiffer_oppdaget* = true **or** $i = n + 1$

- Hvis alle tegn er sifre vil algoritmen terminere når verdien til i er $n + 1$.
- Nå gjenstår Input og Output.

Utkast 3

1. Input n
2. Input $c_1 c_2 \dots c_n$
3. $i \leftarrow 1$; *ikkessiffer_oppdaget* \leftarrow false
4. **Repeat**
 - 4.1. **If** c_i ikke er et siffer **then**
 - 4.1.1. *ikkessiffer_oppdaget* \leftarrow true
 - 4.2. $i \leftarrow i + 1$

until *ikkessiffer_oppdaget* = true **or** $i = n + 1$
5. **If** *ikkessiffer_oppdaget* = true **then**
 - 5.1. Output 'Strengen inneholder tegn som ikke er sifre.'
- else**
 - 5.2. Output 'Strengen inneholder kun sifre.'

- Men, hva hvis $n = 0$? Da vil steg 4.1 ikke kunne utføres.

Kommentarer

- **Repeat-until**-løkken blir alltid utført minst én gang.
- Et alternativ er å bruke en **While**-løkke.
- Da utføres testen i begynnelsen i stedet for på slutten, som med **Repeat-until**.

Løsning

1. Input n

2. Input $c_1 c_2 \dots c_n$
3. $i \leftarrow 0$; $ikkesiffer_oppdaget \leftarrow \text{false}$
4. **While** $ikkesiffer_oppdaget = \text{false}$ **and** $i < n$ **do**
 - 4.1. $i \leftarrow i + 1$
 - 4.2. **If** c_i ikke er et siffer **then**
 - 4.2.1. $ikkesiffer_oppdaget \leftarrow \text{true}$
5. **If** $ikkesiffer_oppdaget = \text{true}$ **then**
 - 5.1. Output 'Strengen inneholder tegn som ikke er sifre.'**else**
 - 5.2. Output 'Strengen inneholder kun sifre.'

Eksempler fra boka

Eksempel (1.3.1).

Lag en algoritme som regner ut x^n , hvor x er et reellt tall og n er et positivt heltall. (Vi antar at vi har multiplikasjon, men ikke eksponensiering.)

Løsning

1. Input x, n
2. $\text{svar} \leftarrow x$
3. **For** $i = 1$ **to** $n - 1$ **do**
 - 3.1. $\text{svar} \leftarrow \text{svar} \times x$
4. Output svar

Kjøring

Steg	x	n	i	svar
1	2	3		
2	2	3	2	
3	2	3	1	2
3.1	2	3	1	4
3	2	3	2	4
3.1	2	3	2	8
4	2	3		8

Eksempel (1.3.2).

Lag en algoritme som bytter verdien til to variable.

Feil

1. $x \leftarrow y$
2. $y \leftarrow x$

Riktig

1. $temp \leftarrow x$
2. $x \leftarrow y$
3. $y \leftarrow temp$

Kjøring

x	y	\rightsquigarrow	x	y
2	3		3	3

Kjøring

Steg	x	y	$temp$
0	2	3	
1	2	3	2
2	3	3	2
3	3	2	2

Eksempel (1.3.3).

Søkere til en jobb tar en test med 20 spørsmål. Lag en algoritme som returnerer en liste av jobbsøkere (identifisert med tall), deres poengsum og en beskjed om hvorvidt de vurderes for stillingen (de som har mer enn 16 poeng) eller settes på venteliste (de med poengsum fra 12 til 15).

- Vi bruker *antall_søkere* for å betegne antall søkere.
- Vi bruker en **For-do**-løkke, fra 1 til *antall_søkere*, og gjør det samme for hver søker.
- De 20 svarene som søker i har avgitt betegner vi med $a_{i,1}, a_{i,2}, \dots, a_{i,20}$.
- Det riktige svaret på spørsmål q betegner vi med c_q .
- Vi bruker enda en **For-do**-løkke for å sjekke svarene.

Løsning

1. Input *antall_søkere*
2. **For** $i = 1$ **to** *antall_søkere* **do**
 - 2.1. $score \leftarrow 0$
 - 2.2. **For** $q = 1$ **to** 20 **do**
 - 2.2.1. Input $a_{i,q}$ [$a_{i,q}$ er søker is svar på spørsmål q .]
 - 2.2.2. **If** $a_{i,q} = c_q$ **then**
 - 2.2.2.1. $score \leftarrow score + 1$
 - 2.3. Output $i, score$
 - 2.4. **If** $score \geq 16$ **then**

2.4.1. Output 'Anbefalt' **else if** score \geq 12 **then**
2.4.2. Output 'Venteliste'

En praktisk forkortelse

Pseudokode

1. **If** $x > 0$ **then**
 - 1.1. Output 'Større'
- else**
 - 1.2. **If** $x < 0$ **then**
 - 1.2.1. Output 'Mindre'

Forkortet pseudokode

1. **If** $x > 0$ **then**
 - 1.1. Output 'Større'
- else if** $x < 0$ **then**
 - 1.2. Output 'Mindre'

Plenumsregning 2

Ukeoppgaver fra kapittel 1 & 2

Roger Antonsen - 24. januar 2008

Oppgave 1.1

Modifiser algoritmen fra 1.2.1 slik at den også returnerer posisjonen i listen hvor det minste tallet forekommer.

Løsning

1. Input the number of values n
2. Input the list of numbers x_1, \dots, x_n
3. $\text{min} \leftarrow x_1$
4. posisjon $\leftarrow 1$
5. **For** $i = 2$ **to** n **do**
 - 5.1. **If** $x_i < \text{min}$ **then**
 - 5.1.1. $\text{min} \leftarrow x_i$
 - 5.1.2. posisjon $\leftarrow i$
6. Output $\text{min}, \text{posisjon}$

Oppgave 1.3

Skriv en algoritme som tar som input et tall n og som regner ut

$$1^2 + 2^2 + 3^2 + \dots + n^2.$$

Løsning

1. Input n
2. $\text{sum} \leftarrow 0$
3. **For** $i = 1$ **to** n **do**
 - 3.1. $\text{sum} \leftarrow \text{sum} + i^2$
4. Output sum

Oppgave 1.6

Skriv en algoritme som tar en liste av tall, som sjekker om tallenes størrelse øker og som returnerer en passende melding. Algoritmen skal designes slik at sjekkingen stopper med en gang svaret er gitt.

Løsning

1. Input x_1, \dots, x_n [$n \geq 1$]
2. $i \leftarrow 1$
3. stigende \leftarrow true
4. **While** $i < n$ **and** stigende **do**
 - 4.1. **If** $x_i > x_{i+1}$ **then**
 - 4.1.1. stigende \leftarrow false
 - 4.2. $i \leftarrow i + 1$
5. **If** stigende **then**
 - 5.1. Output 'Tallene er i stigende rekkefølge'
- else**
 - 5.1. Output 'Tallene er ikke i stigende rekkefølge'

Oppgave 1.9

1. Input n [$n \geq 0$]
 2. $i \leftarrow 0$
 3. **While** n er partall **do**
 - 3.1. $n \leftarrow n/2$
 - 3.2. $i \leftarrow i + 1$
 4. Output i
- (a) Hva returnerer algoritmen når 12 er input?
(b) Hva returnerer algoritmen når input er et oddetall?
(c) Hva skjer når 0 er input?
(d) Er denne sekvensen av steg en algoritme?

Løsning (a)

Steg	n	i
1	12	–
1	12	0
3	12	0
3.1	6	0
3.2	6	1
3	6	1
3.1	3	1
3.2	3	2
3	3	2
4	3	2

Svar (a): 2

Løsning (b)

Steg	n	i
1	oddetall	–
1	oddetall	0
3	oddetall	0
4	oddetall	0

Svar (b): 0

Løsning (c og d)

Steg	n	i
1	0	–
2	0	0
3	0	0
3.1	0	0
3.2	0	1
3	0	1
3.1	0	1
3.2	0	2
3	0	2
3.1	0	2
3.2	0	3
	⋮	

Svar (c): Den terminerer ikke.

Svar (d): Nei.

Oppgave 1.10

1. Input n [$n \geq 0$]
2. $\text{answer} \leftarrow n$
3. **While** $n > 1$ **do**
 - 3.1. $n \leftarrow n - 1$
 - 3.2. $\text{answer} \leftarrow \text{answer} \times n$
4. Output answer

(a) Lag en kjøringstabell som viser hva som skjer når 4 er input.

(b) Er denne sekvensen av steg en algoritme? Begrunn svaret.

Løsning (a)

Steg	n	answer
1	4	—
2	4	4
3	4	4
3.1	3	4
3.2	3	12
3	3	12
3.1	2	12
3.2	2	24
3	2	24
3.1	1	24
3.2	1	24
4	1	24

Løsning (b)

Ja, dette er en algoritme. Stegene er tydelig og utvetydig definert. Sekvensen av steg er veldefinert og vil alltid terminere, siden steg 3 aldri blir utført mer enn $n - 1$ ganger.

Oppgave 1.12

1. Input n [$n \geq 0$]
2. For $i = 1$ to n do
 - 2.1. $a_i \leftarrow 0$
3. For $i = 1$ to n do
 - 3.1. For $j = 1$ to n do
 - 3.1.1. If j kan deles på i then
 - 3.1.1.1. $a_j \leftarrow 1 - a_j$
[a_j er alltid enten 0 eller 1]
4. For $i = 1$ to n do
 - 4.1. Output a_i

Løsning (a)

- | | | | |
|---------|------------|----------|------------|
| 1. gang | 1111111111 | 6. gang | 1001001101 |
| 2. gang | 1010101010 | 7. gang | 1001000101 |
| 3. gang | 1000111000 | 8. gang | 1001000001 |
| 4. gang | 1001111100 | 9. gang | 1001000011 |
| 5. gang | 1001011101 | 10. gang | 1001000010 |

Løsning (b)

$a_j = 1$ nøyaktig når j er et kvadrattall. Grunnen er at det er kun kvadrattall som har et odde antall divisorer.

Oppgave 2.2

Overfør følgende binære tall til desimaltall ved å først skrive dem på ekspandert form.

- (a) 1100101_2
- (b) $1010111, 1011_2$

Løsning

(a) $1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 64 + 32 + 4 + 1 = 101$

(b) $1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = 64 + 16 + 4 + 2 + 1 + \frac{1}{2} + \frac{1}{8} + \frac{1}{16} = 87,6875$

Algoritmen fra forelesningen gir for (a): $0 \xrightarrow{\cdot 2+1} 1 \xrightarrow{\cdot 2+1} 3 \xrightarrow{\cdot 2+0} 6 \xrightarrow{\cdot 2+0} 12 \xrightarrow{\cdot 2+1} 25 \xrightarrow{\cdot 2+0} 50 \xrightarrow{\cdot 2+1} 101$

Oppgave 2.3

Overfør følgende tall fra desimal til binær form.

- (a) 826_{10}
- (b) $0,34375_{10}$
- (c) $1604,175_{10}$
- (d) $-471,25_{10}$

Repetisjon fra boka

- $n \text{ div } 2$ er kvotienten når n deles med 2
- $n \text{ mod } 2$ er resten når n er deles med 2

Eksempel.

$$\begin{aligned} 12 \text{ div } 2 &= 6 \\ 12 \text{ mod } 2 &= 0 \end{aligned}$$

$$\begin{aligned} 13 \text{ div } 2 &= 6 \\ 13 \text{ mod } 2 &= 1 \end{aligned}$$

Algoritme for å overføre fra desimal til binær form (baklengs)

1. Input n [n et naturlig tall]
2. **Repeat**
 - 2.1. Output $n \text{ mod } 2$
 - 2.2. $n \leftarrow n \text{ div } 2$
- until** $n = 0$

Eksempel.

$$\begin{array}{r} 2 \ 8 \\ 4 \ 0 \\ 2 \ 0 \\ 1 \ 0 \\ 0 \ 1 \\ \hline 8_{10} = 1000_2 \end{array}$$

$$\begin{array}{r} 2 \ 7 \\ 3 \ 1 \\ 1 \ 1 \\ 0 \ 1 \\ \hline 7_{10} = 111_2 \end{array}$$

Løsning (a) (826_{10})

$$\begin{array}{r} 2 \ 826 \\ 413 \ 0 \\ 206 \ 1 \\ 103 \ 0 \\ 51 \ 1 \\ 25 \ 1 \\ 12 \ 1 \\ 6 \ 0 \\ 3 \ 0 \\ 1 \ 1 \\ 0 \ 1 \\ \hline 826_{10} = 1100111010_2 \end{array}$$

Mer repetisjon fra boka

- Anta at vi vil finne den binære formen til $0,375_{10}$.
- Siden $0,375$ er mindre enn $0,5$ vil første bit etter komma være 0 .
- Siden $2 \cdot 0,375$ er mindre enn 1 vil første bit etter komma være 0 .

- $\lfloor n \rfloor$ er heltallsdelen av n
- $\text{frac}(n)$ er n minus heltallsdelen av n

Første bit i den binære formen til n er heltallsdelen av $2n$.

Eksempel.

$$\lfloor 2,7 \rfloor = 2$$
$$\text{frac}(2,7) = 0,7$$

$$0,5_{10} = 0,1_2$$
$$0,25_{10} = 0,01_2$$
$$0,75_{10} = 0,11_2$$

Utkast til algoritme

1. Input n [$0 \leq n \leq 1$]
 2. **Repeat**
 - 2.1. $m \leftarrow 2n$
 - 2.2. Output $\lfloor m \rfloor$
 - 2.3. $n \leftarrow \text{frac}(m)$
- until** $n = 0$

Hva hvis $n = 0$ aldri inntreffer?

Eksempel.

$$\begin{array}{r} ,25 \\ 0,5 \\ 1,0 \\ \hline 0,25_{10} = 0,01_2 \end{array}$$

$$\begin{array}{r} ,375 \\ 0,75 \\ 1,5 \\ 1,0 \\ \hline 0,375_{10} = 0,011_2 \end{array}$$

- Vi kan velge hvor nøyaktig vi vil ha svaret.

Algoritme

1. Input n , sifre [$0 \leq n \leq 1$, sifre ≥ 1 , sifre heltall]
2. $i \leftarrow 0$
3. **Repeat**
 - 3.1. $i \leftarrow i + 1$
 - 3.2. $m \leftarrow 2n$

3.3. Output $\lfloor m \rfloor$

3.4. $n \leftarrow \text{frac}(m)$

until $n = 0$ or $i = \text{sifre}$

Løsning (b) $(0,34375_{10})$

$$\begin{array}{r} 34375 \quad 2 \\ 0 6875 \\ 1 375 \\ 0 75 \\ 1 5 \\ 1 0 \\ \hline 0,34375_{10} = 0,01011_2 \end{array}$$

Løsning (c) $(1604,175_{10})$

$$\begin{array}{r} 2 \ 1604 \\ 802 \ 0 \\ 401 \ 0 \\ 200 \ 1 \\ 100 \ 0 \\ 50 \ 0 \\ 25 \ 0 \\ 12 \ 1 \\ 6 \ 0 \\ 3 \ 0 \\ 1 \ 1 \\ 0 \ 1 \\ \hline 1604_{10} = 11001000100_2 \end{array}$$

$$\begin{array}{r} 1875 \quad 2 \\ 0 375 \\ 0 75 \\ 1 5 \\ 1 0 \\ \hline 0,1875_{10} = 0,0011_2 \end{array}$$

Svar: $11001000100,0011_2$

Løsning (d) ($-471, 25_{10}$)

2	471		
	235		,25
	117		2
	58		0
	29		,5
	14		1
	7		,0
	3		-----
	1		$0,25_{10} = 0,01_2$
	0		
	$471_{10} = 111010111_2$		

Svar: $-111010111,01_2$ **Oppgave 2.4**

Overfør følgende tall fra desimal til binær form, med 5 siffer etter komma.

- (a) $0,2_{10}$
- (b) $13,47_{10}$

Løsning (a)

			,2	2
	0		,4	
	0		,8	
	1		,6	
	1		,2	
	0		,4	
			$0,2_{10} = 0,00110_2$	

Oppgave 2.4

Overfør følgende tall fra desimal til binær form, med 5 siffer etter komma.

- (a) $0,2_{10}$
- (b) $13,47_{10}$

Løsning (b)

$$\begin{array}{r}
 2 \ 13 \\
 6 \ 1 \\
 3 \ 0 \\
 1 \ 1 \\
 0 \ 1 \\
 \hline
 13_{10} = 1101_2
 \end{array}$$

$$\begin{array}{r}
 \ ,47 \ 2 \\
 0 \ ,94 \\
 1 \ ,88 \\
 1 \ ,76 \\
 1 \ ,52 \\
 1 \ ,04 \\
 \hline
 0,2_{10} = 0,01111_2
 \end{array}$$

Svar: $1101,01111_2$

Oppgave 2.5

Hva er resultatet på verdien til et naturlig tall hvis

- (a) 0 legges til dets binære representasjon?
- (b) 1 legges til dets binære representasjon?

Løsning

- (a) Tallet blir fordoblet.
- (b) Tallet blir fordoblet og økt med 1.

Plenumsregning 3

Ukeoppgaver fra kapittel 2 & 3

Roger Antonsen - 31. januar 2008

Oppgave 2.7 - Horners metode

(a) 7216_8 : $7 \xrightarrow{8+2} 58 \xrightarrow{8+1} 465 \xrightarrow{8+6} 3726$. Svar: 3726_{10}

(b) 543517_8

(c) $8CB2_{16}$: $8 \xrightarrow{16+12} 140 \xrightarrow{16+11} 2251 \xrightarrow{16+2} 36018$. Svar: 36018_{10}

(d) $E490DF_{16}$

Oppgave 2.8

Skriv ned Horners metode som pseudokode. Anta at basen blir gitt som første argument og deretter at tallet som skal overføres til desimal form blir gitt som en liste med sifre.

Løsning

1. Input base
2. Input x_1, \dots, x_n
3. resultat $\leftarrow x_1$
4. **For** $i = 2$ **to** n **do**
 - 4.1. resultat \leftarrow (resultat \cdot base) $+$ x_i
5. Output resultat

Oppgave 2.14

Utfør følgende regnestykker i binær aritmetikk:

- (a) $1101101_2 + 1011110_2$
- (b) $1001101_2 + 101011_2$
- (c) $1110011_2 - 101101_2$

- (d) $1100010_2 - 1010111_2$
- (e) $10011_2 \cdot 1101_2$
- (f) $11010_2 \cdot 10101_2$
- (g) $110110_2 \div 1001_2$
- (h) $10110_2 \div 11_2$ (3 siffer etter komma)

Løsning

Gjennomgått på tavlen.

Oppgave 3.1

Finn toerkomplementet til følgende 8-bits binære tall.

- (a) 11010100_2
- (b) 01101001_2

Løsning

- (a) $11010100_2 \rightsquigarrow 00101100_2$
- (b) $01101001_2 \rightsquigarrow 10010111_2$

Oppgave 3.6

Hvor mange bit brukes for å lagre $2,8 \cdot 10^{14}$ heltall (ikke-negative heltall lagret uten fortegn-bit)?

Løsning

Hvis vi har n bit, så kan vi lagre 2^n heltall. Vi må derfor finne n slik at $2^n = 2,8 \cdot 10^{14}$, det vil si $n = \log_2(2,8 \cdot 10^{14}) = 47.99242_{10} \approx 48$

Oppgave 3.9

Uttrykk $1101110100,1001_2$ i normalisert binær form, og finn dens 32-bit-datarepresentasjon. Anta at 8 bit brukes for å lagre eksponenten og at eksponentbiasen er $2^7 - 1 = 127$.

Løsning

- $1101110100,1001_2 = 0,11011101001001_2 \cdot 2^{10}$
- Fortegnsbitet er 0, siden tallet er positivt.
- Siden bias er 127, blir eksponentdelen 8-bit-representasjonen av $127_{10} + 10_{10}$. Det vil si $01111111_2 + 1010_2$ eller $10000000_2 + 1001_2$, som er 10001001_2 .
- Signifikanden (mantissen) er $11011101001001\ 000000000$
- Svar: $0 \underbrace{10001001}_{\text{eksp.delen}} \underbrace{11101110\ 10010010\ 00000000}_{\text{signifikanden}}$

Oppgave 3.10

Finn 32-bit-datarepresentasjonen av følgende tall. Anta at 8 bit brukes for å lagre eksponenten og at eksponentbiasen er $2^7 - 1 = 127$.

- (a) $5894,376$
(b) $-0,0387$

Løsning (a: $5894,376$)

- $5894,376_{10} = 1011100000110,0110000001_2 = 0,1011100000110\ 0110000001 \cdot 2^{13}$
- Eksponentdelen er $127_{10} + 13_{10} = 01111111_2 + 1101_2 = 10001100_2$.
- Signifikanden er $1011100\ 00011001\ 10000001$
- Svar: $0 \underbrace{10001100}_{\text{eksp.delen}} \underbrace{01011100\ 00011001\ 10000001}_{\text{signifikanden}}$

Løsning (b: $-0,0387$)

- Fortegnsbitet er 1.

- $0,0387_{10} = 0,0000\ 1001111\ 01000001\ 11110010_2 = 0,1001111\ 01000001\ 11110010_2 \cdot 2^{-4}$
- Eksponentdelen er $127_{10} + (-4_{10}) = 01111111_2 + (-0100_2) = 01111011_2$.
- Signifikanden er $1001111\ 01000001\ 11110010$.
- Svar: $1 \underbrace{01111011}_\text{eksp.delen} \underbrace{11001111\ 01000001\ 11110010}_\text{signifikanden}$

Oppgave 3.11

Gjenta oppgave 3.9 for en 32-bit datamaskin hvor 12 bit brukes til eksponenten og eksponentbiasen er $2^{11} - 1 = 2047$.

Løsning

- $1101110100,1001_2 = 0,11011101001001_2 \cdot 2^{10}$
- Fortegnsbitet er 0.
- Siden bias er 2047, blir eksponentdelen 12-bit-representasjonen av $2047_{10} + 10_{10}$. Det vil si $0111\ 1111\ 1111_2 + 1010_2$ eller $1000\ 0000\ 0000_2 + 1001_2$, som er $1000\ 0000\ 1001_2$.
- Signifikanden er på $32 - 13 = 19$ bit: $11011101001001\ 00000$
- Svar: $0 \underbrace{1000000\ 0100\ 1}_\text{eksp.delen: 12 bit} \underbrace{1110\ 11101001\ 00100000}_\text{signifikanden: 19 bit}$

Oppgave 3.12

Finn, på desimalform, det omtrentlige området av positive reelle tall som kan representeres med 64 bit, hvor 11 bit brukes til eksponentdelen og eksponentbiasen er $2^{10} - 1 = 1023$

Løsning

- Eksponentdelen tar verdier fra 0 til $2^{11} - 1 = 2047$, så eksponentene som kan representeres går fra -1023 (som representeres med 11 nullere) til 1024 (som representeres med 11 enere).
- Signifikanden m er slik at $0,1_2 \leq m < 1_2$.
- Det minste tallet som kan representeres er derfor $0,1_2 \cdot 2^{-1023}$
- En øvre grense for hva som kan representeres er $1_2 \cdot 2^{1024}$.

- Hva om vi ønsker å uttrykke dette på normalisert form med base 10, det vil si på formen $m \cdot 10^e$, hvor e er et heltall og $0,1_{10} \leq m < 1_{10}$?

Løsning

- Vi ser først på $0,1_2 \cdot 2^{-1023} = 2^{-1024}$ og finner x slik at $10^x = 2^{-1024}$.

$$\begin{aligned}
 x &= \log_{10}(2^{-1024}) \\
 &= -1024 \cdot \log_{10}(2) \\
 &= -308,25472 \\
 10^x &= 10^{-308,25472} \\
 &= 10^{-0,25472} \cdot 10^{-308} \\
 &= 0,55626278 \cdot 10^{-308} \\
 &\approx 0,56 \cdot 10^{-308}
 \end{aligned}$$

- Det minste tallet som kan representeres er $0,56 \cdot 10^{-308}$.

Løsning

- Vi ser så på $1_2 \cdot 2^{1024}$ og finner x slik at $10^x = 2^{1024}$.

$$\begin{aligned}
 x &= \log_{10}(2^{1024}) \\
 &= 1024 \cdot \log_{10}(2) \\
 &= 308,25472 \\
 10^x &= 10^{308,25472} \\
 &= 10^{0,25472} \cdot 10^{308} \\
 &= 1,7977115 \cdot 10^{308} \\
 &= 0,17977115 \cdot 10^{309} \\
 &\approx 0,18 \cdot 10^{309}
 \end{aligned}$$

- En øvre grense for hva som kan representeres er $0,18 \cdot 10^{309}$.

Oppgave 3.13

Meningen med følgende algoritme er at den skal skrive ut kubene av tall fra 0, 1 til 10 i steg på 0, 1. Forklar problemet som kan oppstå når algoritmen implementeres på en datamaskin.

1. $x \leftarrow 0,0$
 2. **Repeat**
 - 2.1. $x \leftarrow x + 0,1$
 - 2.2. $x_cubed \leftarrow x^3$
 - 2.3. Output x, x_cubed
- until** $x = 10,0$

Løsning

Problemet ligger i at reelle tall ikke representeres nøyaktig og at vi ikke har noen garanti for at betingelsen $x = 10,0$ vil være oppfylt. En løsning er å erstatte $x = 10,0$ med $|x - 10,0| < 0,05$.

Oppgave 1, side 11, forelesningen 23.01.08

Finn en pseudokode for å beregne toerkomplementet til en sekvens av lengde 8, av lengde 16 og av lengde 32.

Løsning

Vi skisserer først en idé og skriver deretter pseudokode.

- Vi må undersøke sekvensen fra høyre og hoppe over alle nullere.
- En **For**-løkke er uegnet, siden vi må avbryte når vi treffer en ener.
- Det er naturlig å bruke en variabel i for å telle. Denne kan settes til lengden på sekvensen ved start.
- Siden første bit fra høyre kan være en ener, og vi da må avbryte, er en **While**-løkke velegnet.
- Når vi er ute av **While**-løkken, må vi sørge for at eneren ikke flippes.
- Deretter skal alle bit til venstre for eneren flippes. Vi må ta høyde for at det ikke trenger å være noen bit igjen, det vil si at i er 0.
- Derfor kan enda en **While**-løkke være passende her.

Løsning

1. Input x_1, x_2, \dots, x_n [n lik 8, 16 eller 32]
2. $i \leftarrow n$
3. **While** $1 \leq i$ **and** $x_i = 0$ **do**
 - 3.1. $i \leftarrow i - 1$
4. $i \leftarrow i - 1$
5. **While** $1 \leq i$ **do**
 - 5.1. $x_i \leftarrow 1 - x_i$
 - 5.2. $i \leftarrow i - 1$
6. Output x_1, \dots, x_n

Oppgave 2, side 11, forelesningen 23.01.08

Hvis vi tar toerkomplementet av toerkomplementet av en sekvens $x_1 \cdots x_k$ av nuller og enere, får vi den opprinnelige sekvensen tilbake. Forklar hvorfor.

Løsning

Anta at den første eneren fra høyre fins på posisjon n , hvor $1 \leq n \leq k$. Når vi tar toerkomplementet, så vil x_1 til x_{n-1} flippes og x_n til x_k forbli urørt. Når vi tar toerkomplementet en gang til skjer det samme igjen: x_1 til x_{n-1} flippes og x_n til x_k forblir urørt. Siden x_1 til x_{n-1} blir flippet to ganger, får vi tilbake utgangspunktet.

$$\begin{array}{l} \text{toerkomplementet:} \\ \text{toerkomplementet:} \end{array} \begin{array}{l} x_1 x_2 x_3 \dots x_{n-1} x_n x_{n+1} x_{n+2} \dots x_k \\ \bar{x}_1 \bar{x}_2 \bar{x}_3 \dots \bar{x}_{n-1} x_n x_{n+1} x_{n+2} \dots x_k \\ x_1 x_2 x_3 \dots x_{n-1} x_n x_{n+1} x_{n+2} \dots x_k \end{array}$$

Plenumsregning 4

Ukeoppgaver fra kapittel 3 & 4

Roger Antonsen - 7. februar 2008

Oppgave 3.15

Forklar følgende påstand ved å vise til beregninger med reelle tall på eksponentiell form: "Man mister presisjon når nesten like tall subtraheres."

Løsning

Man mister presisjon fordi antall gjeldende/signifikante siffer reduseres. Eksempel:

$$\begin{array}{r} 0,23456 \cdot 10^5 \quad 5 \text{ gjeldende siffer} \\ - 0,2345 \cdot 10^5 \quad 4 \text{ gjeldende siffer} \\ \hline = 0,00006 \cdot 10^5 \\ = 0,6 \cdot 10^1 \quad 1 \text{ gjeldende siffer} \end{array}$$

Oppgave 4.1

Uttrykk følgende utsagn i utsagnslogikk og identifiser hovedkonnektivene.

- (a) Either Karen is studying computing and Minh is not studying mathematics, or Minh is studying mathematics.
- (b) It is not the case that if it is sunny then I will carry an umbrella.
- (c) The program will terminate if and only if the input is not numeric or the escape key is pressed.
- (d) If $x = 7$ and $y \neq 4$ and $z = 2$, then if it is not true that either $y = 4$ or $z \neq 2$ then $x = 7$ or $z = 2$.

Løsning 4.1

- (a) Either Karen is studying computing and Minh is not studying mathematics, or Minh is studying mathematics.

p: "Karen is studying computing"
q: "Minh is studying mathematics"

Svar: $(p \wedge \neg q) \vee q$

Hovedkonnektivet er \vee (eller)

(b) It is not the case that if it is sunny then I will carry an umbrella.

p: "it is sunny" q: "I will carry an umbrella"

Svar: $\neg(p \rightarrow q)$

Hovedkonnektivet er \neg (ikke)

Løsning 4.1

(c) The program will terminate if and only if the input is not numeric or the escape key is pressed.

p: "the program will terminate"

q: "the input is numeric"

r: "the escape key is pressed"

Svar: $p \leftrightarrow (\neg q \vee r)$

Hovedkonnektivet er \leftrightarrow (hvis-og-bare-hvis)

(d) If $x = 7$ and $y \neq 4$ and $z = 2$, then if it is not true that either $y = 4$ or $z \neq 2$ then $x = 7$ or $z = 2$.

p: " $x = 7$ " q: " $y = 4$ " r: " $z = 2$ "

Svar: $(p \wedge \neg q \wedge r) \rightarrow (\neg(q \vee \neg r) \rightarrow (p \vee r))$

Hovedkonnektivet er første forekomst av \rightarrow (hvis-så)

Oppgave 4.2

p: det snør

q: jeg skal gå på ski

Løsning

(a) $\neg p \wedge q$:

Det snør ikke og jeg skal gå på ski.

(b) $p \rightarrow q$:

Hvis det snør, så skal jeg gå på ski.

(c) $\neg q \rightarrow p$:

Hvis jeg ikke skal gå på ski, så snør det.

(d) $(p \vee \neg q) \wedge p$:

Enten snør det eller så skal jeg ikke gå på ski, og det snør.

Oppgave 4.3

- (a) Skriv opp sannhetsverditabellen for konnektivet **xor** med symbolet \oplus , hvor $p \oplus q$ betyr "Enten p eller q , men ikke begge."
- (b) Skriv opp sannhetsverditabeller som viser at $p \oplus q$ er logisk ekvivalent med $(p \vee q) \wedge \neg(p \wedge q)$.

Løsning

p	q	$(p \oplus q)$	$(p \vee q)$	$(p \wedge q)$	$\neg(p \wedge q)$	$(p \vee q) \wedge \neg(p \wedge q)$
T	T	F	T	T	F	F
T	F	T	T	F	T	T
F	T	T	T	F	T	T
F	F	F	F	F	T	F

Kolonne 1–3 er svar på (a). For (b), så må det påpekes at kolonne 3 og 7 er like.

Oppgave 4.6

Sett opp sannhetsverditabeller for følgende uttrykk. For hvert tilfelle, finn ut om det er en tautologi, en kontradiksjon eller ingen av delene.

Løsning

Vi gjør (a) og (d) her, og eventuelt (b) og (c) på tavlen.

Løsning

- (a) $\neg(p \vee \neg q) \vee p$

p	q	$\neg(p \vee \neg q) \vee p$
T	T	T
T	F	T
F	T	F
F	F	T

- Utsagnet er hverken en tautologi eller en kontradiksjon.

Løsning

(d) $[(p \wedge r) \vee (q \wedge r)] \rightarrow (p \rightarrow \neg q)$

p	q	r	$[(p \wedge r) \vee (q \wedge r)]$	$(p \rightarrow \neg q)$
T	T	T	T	F
T	T	F	F	T
T	F	T	T	T
T	F	F	F	T
F	T	T	F	T
F	T	F	F	T
F	F	T	F	T
F	F	F	F	T

- Utsagnet er hverken en tautologi eller en kontradiksjon.

Oppgave 4.11

If ikke($x \geq 3$ og $x < 6$) then ...

Løsning

- ikke($x \geq 3$ og $x < 6$) kan skrives om til
- (ikke $x \geq 3$) eller (ikke $x < 6$), som kan skrives om til
- $x < 3$ eller $x \geq 6$
- Vi får da: **If** $x < 3$ eller $x \geq 6$ **then** ...

Oppgave 4.12

Skriv om følgende pseudokode ved å erstate **Repeat-until**-løkken med en **While-do**-løkke:

1. $n \leftarrow 0$
2. $\text{term} \leftarrow 1$

```
3. sum ← 0
4. Repeat
  4.1. n ← n + 1
  4.2. term ← term/2
  4.3. sum ← sum + term
until term < 0,001 eller n = 100
```

Løsning

- Repeat-until-løkken blir utført minst én gang.
- Så lenge betingelsen ($\text{term} < 0,001$ eller $n = 100$) *ikke* er oppfylt, så skal koden i løkken utføres. Vi kan derfor se på betingelsen “ikke ($\text{term} < 0,001$ eller $n = 100$)”:
 - ikke ($\text{term} < 0,001$ eller $n = 100$)
 - (ikke $\text{term} < 0,001$) og (ikke $n = 100$)
 - $\text{term} \geq 0,001$ og $n \neq 100$

Løsning

Gammel kode

```
1. n ← 0
2. term ← 1
3. sum ← 0
4. Repeat
  4.1. n ← n + 1
  4.2. term ← term/2
  4.3. sum ← sum + term
until term < 0,001 eller n = 100
```

Ny kode

```
1. n ← 0
2. term ← 1
3. sum ← 0
4. n ← n + 1
5. term ← term/2
6. sum ← sum + term
7. While term  $\geq$  0,001 og n  $\neq$  100 do
  7.1. n ← n + 1
  7.2. term ← term/2
  7.3. sum ← sum + term
```

Oppgave 4.16

Finn et uttrykk som er logisk ekvivalent med $p \vee q$, men som kun bruker konnektivene \wedge og \neg .

Løsning

- Vi vet at $p \vee q$ er logisk ekvivalent med $\neg\neg(p \vee q)$.
- Vi vet også at $\neg(p \vee q)$ er logisk ekvivalent med $\neg p \wedge \neg q$.
- Da må $\neg\neg(p \vee q)$, og $p \vee q$, være logisk ekvivalent med $\neg(\neg p \wedge \neg q)$.

p	q	$p \vee q$	\neg	$(\neg p \wedge \neg q)$
T	T	T	T	F
T	F	T	T	F
F	T	T	T	F
F	F	F	F	T

Oppgave 4.17

p	q	$p q$
T	T	F
T	F	T
F	T	T
F	F	T

- Finn et uttrykk som er logisk ekvivalent med $\neg p$ som kun bruker $|$.
- Finn et uttrykk som er logisk ekvivalent med $p \wedge q$ som kun bruker $|$.
- Finn et uttrykk som er logisk ekvivalent med $p \vee q$ som kun bruker $|$.

Løsning

	p	$p p$
(a)	T	F
	F	T

	p	q	$p \wedge q$	$p q$	$(p q) (p q)$
(b)	T	T	T	F	T
	T	F	F	T	F
	F	T	F	T	F
	F	F	F	T	F

	p	q	$p \vee q$	$p p$	$q q$	$(p p) (q q)$
(c)	T	T	T	F	F	T
	T	F	T	F	T	T
	F	T	T	T	F	T
	F	F	F	T	T	F

Oppgave 4.24

Denne setningen har fem ord.

- (a) Hva er sannhetsverdien til denne påstanden?
- (b) Skriv ned negasjonen til påstanden. Hva er sannhetsverdien?

Løsning

- (a) Den er sann.
- (b) "Denne setningen har ikke fem ord." Den er også sann. (Diskusjon i plenum.)

Plenumsregning 5

Ukeoppgaver fra kapittel 4

Roger Antonsen - 14. februar 2008

Oppgave 4.4

Skriv ned setninger som svarer til den konverse og den kontrapositive av følgende utsagn.

Husk at hvis $p \rightarrow q$ er påstanden, så har vi at

- $q \rightarrow p$ er den konverse, og at
- $\neg q \rightarrow \neg p$ er den kontrapositive.

Den konverse betyr noe annet enn den opprinnelige påstanden, mens den kontrapositive er logisk ekvivalent med den opprinnelige påstanden.

Løsning

(a) Hvis inputfilen eksisterer, så genereres ikke en feilmelding.

- Konvers:
Hvis det ikke genereres en feilmelding, så eksisterer inputfilen.
- Kontrapositiv:
Hvis det genereres en feilmelding, så eksisterer ikke inputfilen.

Løsning

(b) Hvis databasen ikke er tilgjengelig, så kan ikke programmet mitt kjøre.

- Konvers:
Hvis ikke programmet mitt kan kjøre, så er ikke databasen tilgjengelig.
- Kontrapositiv:
Hvis programmet mitt kan kjøre, så er databasen tilgjengelig.

Løsning

(c) Hvis programmet ikke inneholder noen feil, så gir det korrekt output.

- Konvers:
Hvis programmet gir korrekt output, så inneholder det ikke noen feil.
- Kontrapositiv:
Hvis programmet ikke gir korrekt output, så inneholder det noen feil.

Oppgave 4.7

La P og Q stå for logiske uttrykk. Hvis P er *usann* for en gitt tilordning av sannhetsverdier til variablene, så må $P \wedge Q$ være *usann* for den tilordningen, så det er ikke nødvendig å finne verdien til Q .

- (a) Gi en tilsvarende regel for $P \vee Q$
- (b) Konstruer, ved å bruke disse reglene som snarveier, sannhetsverditabellene for følgende uttrykk.
- (i) $[\neg(p \wedge q) \wedge (p \vee \neg r)] \wedge [(p \wedge r) \vee \neg q]$
- (ii) $\neg[\neg p \wedge (q \vee r)] \vee (\neg p \wedge \neg r)$

Løsning (a)

Hvis P er *sann* for en gitt tilordning av sannhetsverdier til variablene, så må $P \vee Q$ være *sann* for den tilordningen, så det er ikke nødvendig å finne verdien til Q .

Løsning (b - i)

p	q	r	\neg	$(p \wedge q)$	\wedge	$(p \vee \neg r)$	\wedge	$[(p \wedge r) \vee \neg q]$			
T	T	T	F	T	F	–	F	–	–	–	–
T	T	F	F	T	F	–	F	–	–	–	–
T	F	T	T	F	T	T	T	–	T	T	T
T	F	F	T	F	T	T	T	–	T	T	T
F	T	T	T	F	F	F	F	–	–	–	–
F	T	F	T	F	T	T	F	F	F	F	F
F	F	T	T	F	F	F	F	–	–	–	–
F	F	F	T	F	T	T	T	–	T	T	T

Løsning (b - ii)

p	q	r	¬	[¬p	∧	(q ∨ r)]	∨	(¬p ∧ ¬r)
T	T	T	T	F	F	–	T	–
T	T	F	T	F	F	–	T	–
T	F	T	T	F	F	–	T	–
T	F	F	T	F	F	–	T	–
F	T	T	F	T	T	T	F	F
F	T	F	–	–	–	–	T	T
F	F	T	F	T	T	T	F	F
F	F	F	–	–	–	–	T	T

Oppgave 4.8

Bruk sannhetsverditabeller for å vise at $\neg(p \vee \neg q)$ og $\neg p \wedge q$ er logisk ekvivalente.

Løsning

p	q	¬	(p	∨	¬q)	¬p	∧	q
T	T	F	T	T	F	F	F	T
T	F	F	T	T	T	F	F	F
F	T	T	F	F	F	T	T	T
F	F	F	F	T	T	T	F	F

Oppgave 4.9

Bruk sannhetsverditabeller for å vise følgende logiske lover.

(a) $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$

(b) $p \wedge (p \vee q) \equiv p$

Løsning (a)

p	q	r	$p \wedge (q \vee r)$			$(p \wedge q) \vee (p \wedge r)$		
T	T	T	T	T	T	T	T	–
T	T	F	T	T	T	T	T	–
T	F	T	T	T	T	–	T	T
T	F	F	–	F	F	F	F	F
F	T	T	F	F	–	F	F	F
F	T	F	F	F	–	F	F	F
F	F	T	F	F	–	F	F	F
F	F	F	F	F	–	F	F	F

Oppgave 4.18

Uttrykk følgende påstander i predikatlogikk, og finn deres sannhetsverdier.

- (a) Det fins et reellt tall x slik at $x^2 - 3x + 2 = 0$.
- (b) For ethvert reellt tall x , så fins det et reellt tall y slik at $x = y^2$.

Løsning

- (a) $\exists x(x \in \mathbb{R} \wedge x^2 - 3x + 2 = 0)$ eller $\exists x(x^2 - 3x + 2 = 0)$
Sant; hvis vi lar $x = 2$, så ser vi at påstanden holder.
- (b) $\forall x(x \in \mathbb{R} \rightarrow \exists y(y \in \mathbb{R} \wedge x = y^2))$ eller $\forall x \exists y(x = y^2)$
Usant; vi får et moteksempel ved å la $x = -1$.

Oppgave 4.19

Skriv negasjonene til påstandene i oppgave 4.18, både symbolsk og på norsk.

Løsning

- (a) $\neg \exists x(x^2 - 3x + 2 = 0)$ eller $\forall x(x^2 - 3x + 2 \neq 0)$
For alle reelle tall x , så er det ikke slik at $x^2 - 3x + 2 = 0$.
- (b) $\neg \forall x \exists y(x = y^2)$ eller $\exists x \forall y(x \neq y^2)$
Det fins et reellt tall x slik at for alle y , så er det ikke slik at $x = y^2$.

Oppgave 4.20

I spesifikasjonen til et biblioteksystem har vi følgende predikatsymboler:

- $B(p, b)$, som står for predikatet "person p har lånt bok b ", og
- $O(b)$, som står for predikatet "bok b har forfalt".

Utrykk følgende setninger i predikatlogikk.

Løsning

(a) Person p har lånt en bok. (Anta at *en* betyr *minst en*.)

- Det fins en bok x slik at p har lånt x .
- $\exists x(\text{person } p \text{ har lånt bok } x)$
- $\exists xB(p, x)$

(b) Bok b er utlånt.

- Det fins en person x slik at x har lånt bok b .
- $\exists x(\text{person } x \text{ har lånt bok } b)$
- $\exists xB(x, b)$

(c) Bok b står på hylla.

- Det fins ingen som har lånt bok b .
- $\neg\exists xB(x, b)$ eller $\forall x\neg B(x, b)$

(d) Person p har lånt minst to bøker.

- Det fins en bok x og en bok y slik at person p har lånt x og y .
- Vi må også si at x og y ikke kan være samme bok.
- $\exists x\exists y(B(p, x) \wedge B(p, y) \wedge x \neq y)$

Løsning

(e) Ingen bok har blitt lånt av mer enn en person.

- Det fins ikke en bok b som er lånt av to (forskjellige) personer.
- $\neg\exists b(\text{to forskjellige personer har lånt } b)$
- $\neg\exists b\exists x\exists y(B(x, b) \wedge B(y, b) \wedge x \neq y)$
- alternativt: $\forall b\neg\exists x\exists y(B(x, b) \wedge B(y, b) \wedge x \neq y)$
- alternativt: $\forall b\forall x\forall y\neg(B(x, b) \wedge B(y, b) \wedge x \neq y)$
- alternativt: $\forall b\forall x\forall y(B(x, b) \wedge B(y, b) \rightarrow x = y)$

(f) Det fins ingen forfalte bøker.

- Det fins ikke en bok x slik at x har forfalt.
- $\neg\exists xO(x)$ eller $\forall x\neg O(x)$

(g) Hvis en bok har forfalt, så må den være utlånt.

- For alle bøker x , hvis x har forfalt, så fins en person y som har lånt x .
- $\forall x(O(x) \rightarrow \exists yB(y, x))$

(h) Person p har en forfalt bok.

- Det fins en forfalt bok x slik at person p har lånt x .
- $\exists x(O(x) \wedge B(p, x))$

Oppgave 4.21 (a) Bevis følgende påstand

Summen av et partall og et oddetall er et oddetall.

Bevis.

- Anta at x er et partall og at y er et oddetall.
- Siden x er et partall, så fins det et heltall m slik at $x = 2m$.
- Siden y er et oddetall, så fins det et heltall n slik at $y = 2n + 1$.
- Summen $x + y$ blir da $2m + 2n + 1$.
- Ved å faktorisere får vi at $x + y$ er $2(m + n) + 1$.
- Siden $x + y$ er på formen $2s + 1$, så må det være et oddetall.
- Siden x og y var *vilkårlig* valgt, vil påstanden gjelde for *alle* slike tall.

Oppgave 4.21 (b) Bevis følgende påstand

Produktet av to oddetall er et oddetall.

Bevis.

- Anta at x er et oddetall og at y er et oddetall.
- Siden x er et oddetall, så fins det et heltall m slik at $x = 2m + 1$.
- Siden y er et oddetall, så fins det et heltall n slik at $y = 2n + 1$.

- Produktet xy blir da $(2m + 1)(2n + 1)$.
- Ved å gange ut får vi at xy er $4mn + 2m + 2n + 1$.
- Ved å faktorisere får vi at xy er $2(2mn + m + n) + 1$.
- Siden xy er på formen $2s + 1$, så må det være et oddetall.
- Siden x og y var *vilkårlig* valgt, vil påstanden gjelde for *alle* slike tall.

Oppgave 4.21 (c) Bevis følgende påstand

Hvis $x + y < 2$, så $x < 1$ eller $y < 1$, for reelle tall x og y .

Bevis.

- Et indirekte bevis får vi ved å anta det motsatte av påstanden og komme frem til en selvmotsigelse.
- Anta for motsigelse at det fins reelle tall x og y slik at $x + y < 2$ men hverken $x < 1$ eller $y < 1$.
- Siden $x \not< 1$, har vi $x \geq 1$.
- Siden $y \not< 1$, har vi $y \geq 1$.
- Da må $x + y \geq 2$.
- Det gir en motsigelse, siden vi har antatt $x + y < 2$.
- Vi konkluderer med at påstanden holder.

Bevis (Alternativt bevis).

- La x og y være reelle tall slik at $x + y < 2$.
- Hvis $x < 1$, så er vi ferdige.
- Vi kan derfor anta at $x \geq 1$.
- Vi må så vise at $y < 1$.
- Siden $x + y < 2$, så må $y < 2 - x$.
- Siden $y < 2 - x$ og $x \geq 1$, så må $y < 1$, og vi er ferdige.

Oppgave 4.21 (d) Bevis følgende påstand

Summen av fem etterfølgende heltall er delelig med 5.

Bevis.

- La y være summen av fem etterfølgende heltall.
- Anta at $y = x + (x + 1) + (x + 2) + (x + 3) + (x + 4)$.
- Vi må vise at y er delelig med 5.
- Vi får at $y = 5x + 1 + 2 + 3 + 4 = 5x + 10 = 5(x + 2)$.
- Vi konkluderer med at y er delelig med 5.

Oppgave 4.21 (e) Bevis følgende påstand

Hvis n er et heltall, så er $n^2 + n$ et partall.

Bevis.

- n må enten være et partall eller et oddetall.
- Anta først at n er et partall. Da fins det en m slik at $n = 2m$.

$$\begin{aligned}n^2 + n &= (2m)^2 + (2m) \\ &= 4m^2 + 2m \\ &= 2(2m^2 + m)\end{aligned}$$

- Anta så at n er et oddetall. Da fins det en m slik at $n = 2m + 1$.

$$\begin{aligned}n^2 + n &= (2m + 1)^2 + (2m + 1) \\ &= (4m^2 + 4m + 1) + (2m + 1) \\ &= 4m^2 + 6m + 2 \\ &= 2(2m^2 + 3m + 1)\end{aligned}$$

- Siden $n^2 + n$ i begge tilfeller er lik $2x$, for et heltall x , så må $n^2 + n$ være et partall.

Oppgave 4.21 (f) Bevis følgende påstand

Hvis n er et oddetall, så er $n^2 - 1$ delelig med 4.

Bevis.

- Siden n er et oddetall, fins en m slik at $n = 2m + 1$.

$$\begin{aligned}n^2 - 1 &= (2m + 1)^2 - 1 \\ &= (4m^2 + 4m + 1) - 1 \\ &= 4m^2 + 4m \\ &= 4(m^2 + m)\end{aligned}$$

- Siden $n^2 - 1 = 4(m^2 + m)$, så må $n^2 - 1$ være delelig med 4.

Oppgave 4.22

Fyll inn detaljene i bevisskissen for at $\sqrt{2}$ er et irrasjonalt tall.

Løsning

1. Anta at $m/n = \sqrt{2}$, hvor m og n er naturlige tall. Forklar hvorfor vi kan anta at m og n ikke begge er partall.
 - Vi kan alltid forkorte en brøk slik at teller og nevner ikke har noen felles faktorer. Hvis både m og n hadde vært partall, kunne vi ha delt teller og nevner med 2 og fått en enklere brøk. Derfor kan vi anta at m/n allerede er forkortet maksimalt.
2. Utled at $m^2 = 2n^2$, og forklar så hvorfor m må være et partall.
 - Siden $m/n = \sqrt{2}$, så får vi ved å ta kvadratet av hver side $(m/n)^2 = (\sqrt{2})^2$, det vil si $m^2/n^2 = 2$. Ved å gange med n^2 på begge sider, får vi $m^2 = 2n^2$.
 - Siden $m^2 = 2n^2$, så må m^2 være et partall. Vi må vise at m også er et partall. Hvis m hadde vært et oddetall, så måtte m^2 også ha vært et oddetall. (Sjekk selv.) Siden m^2 ikke er et oddetall, så kan ikke m være oddetall heller. Vi konkluderer med at m er et partall.

Løsning

3. La $m = 2k$ (hvor k er et naturlig tall), og utled at n er et partall.
 - Fra $m^2 = 2n^2$ får vi at $(2k)^2 = 4k^2 = 2n^2$, det vil si at $2k^2 = n^2$. Det betyr at n^2 er et partall. Da må n også være et partall (ved samme argument som over).
4. Forklar hvorfor dette viser at $\sqrt{2}$ er irrasjonalt.
 - Fordi vi har kommet frem til en motsigelse ved at både m og n er partall.

- Vi begynte med å anta for motsigelse at $\sqrt{2}$ var rasjonalt, det vil si at $\sqrt{2} = m/n$, hvor m og n er naturlige tall. Vi antok at m/n var maksimalt forkortet og at m og n derfor ikke begge kunne være partall. Deretter utledet vi at både m og n måtte være partall allikevel og fikk en motsigelse.

Oppgave 4.23

Finn et moteksempel til følgende påstander.

Løsning

- (a) Ethvert naturlig tall delelig med både 4 og 6 må også være delelig med 24.
- Et moteksempel er 12.
- (b) Hvis n er et naturlig tall, så må $n^4 + 4$ være et multippel av 5.
- Et moteksempel er 5 eller 10.
- (c) Ethvert naturlig tall kan uttrykkes på formen $x^2 + y^2 + z^2$, for ikke-negative heltall x , y og z .
- Et moteksempel er 7.

Plenumsregning 6

Ukeoppgaver fra kapittel 5

Roger Antonsen - 21. februar 2008

Oppgave 5.1

Skriv følgende mengder på listeform.

- (a) Mengden av alle vokaler
- (b) $\{x \in \mathbb{N} : 10 \leq x \leq 20 \text{ og } x \text{ er delbar med } 3\}$
- (c) Mengden av alle naturlige tall som gir en rest på 1 når de deles på 5.

Løsning

- (a) $\{a, e, i, o, u, y, \text{æ}, \text{ø}, \text{å}\}$
- (b) $\{12, 15, 18\}$
- (c) $\{1, 6, 11, 16, \dots\}$

Oppgave 5.2

Skriv ned følgende mengder på "predikatform".

- (a) $\{4, 8, 12, 16, 20\}$
- (b) $\{000, 001, 010, 011, 100, 101, 110, 111\}$
- (c) $\{1, 4, 9, 16, 25, \dots\}$

Løsning

- (a) $\{x \in \mathbb{N} : x \leq 20 \text{ og } x \text{ er delbar med } 4\}$ eller $\{x : x = 4y \text{ for et naturlig tall } y \leq 5\}$
- (b) $\{x : x \text{ er en streng med tre bit}\}$ eller $\{x : x \in \{0, 1\}^3\}$
- (c) $\{x : \text{det fins et naturlig tall } y \text{ slik at } x = y^2\}$ eller $\{x^2 : x \in \mathbb{N}\}$ eller $\{x \in \mathbb{N} : \exists y(x = y^2)\}$ eller $\{x : x \text{ er et kvadrattall}\}$

Oppgave 5.3

La $A = \{1, \{1\}, \{2\}, 3\}$. Avgjør hvilke av følgende påstander som er sanne og usanne.

Løsning

- | | |
|----------------------------------|-------------------------------|
| (a) $1 \in A$ Sann | (f) $2 \in A$ Usann |
| (b) $1 \subseteq A$ Usann | (g) $\{2\} \in A$ Sann |
| (c) $\{1\} \in A$ Sann | (h) $\{2\} \subseteq A$ Usann |
| (d) $\{1\} \subseteq A$ Sann | (i) $\{3\} \in A$ Usann |
| (e) $\{\{1\}\} \subseteq A$ Sann | (j) $\{3\} \subseteq A$ Sann |

Oppgave 5.4

La

$$\mathcal{E} = \{x \in \mathbb{N} : x \leq 12\},$$

$$A = \{x : x \text{ er oddetall}\},$$

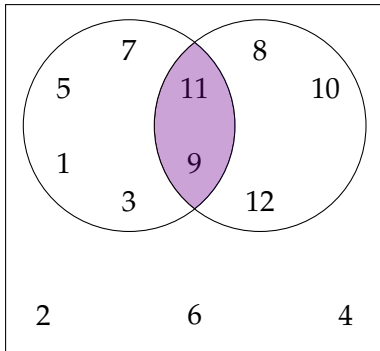
$$B = \{x : x > 7\}, \text{ og}$$

$$C = \{x : x \text{ er delelig med } 3\}.$$

Lag Venn-diagrammer for mengdene. Skriv ned følgende mengder på listeform.

- (a) $A \cap B$
- (b) $B \cup C$
- (c) \bar{A}
- (d) $(A \cup \bar{B}) \cap C$
- (e) $\overline{(A \cup C)} \cup \bar{C}$

- (a) $A \cap B$



$$E = \{x \in \mathbb{N} : x \leq 12\}$$

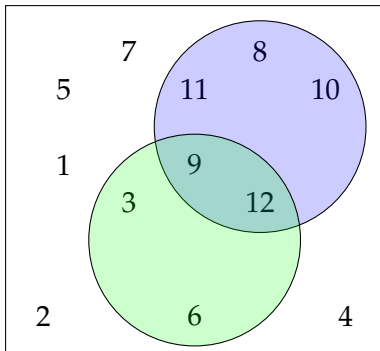
$$A = \{x : x \text{ er oddetall}\}$$

$$B = \{x : x > 7\}$$

$$C = \{x : x \text{ er delelig med } 3\}$$

Svar (a): $\{9, 11\}$

(b) $B \cup C$



$$E = \{x \in \mathbb{N} : x \leq 12\}$$

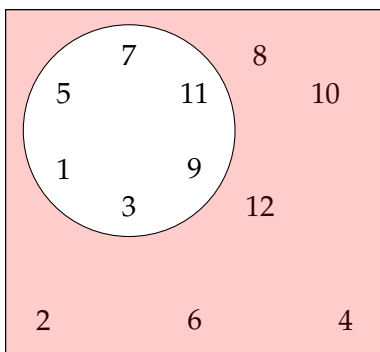
$$A = \{x : x \text{ er oddetall}\}$$

$$B = \{x : x > 7\}$$

$$C = \{x : x \text{ er delelig med } 3\}$$

Svar (b): $\{3, 6, 8, 9, 10, 11, 12\}$

(c) \bar{A}



$$E = \{x \in \mathbb{N} : x \leq 12\}$$

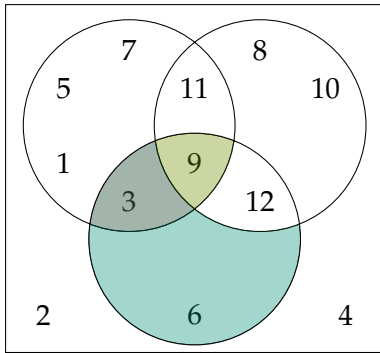
$$A = \{x : x \text{ er oddetall}\}$$

$$B = \{x : x > 7\}$$

$$C = \{x : x \text{ er delelig med } 3\}$$

Svar (c): $\{2, 4, 6, 8, 10, 12\}$

(d) $(A \cup \bar{B}) \cap C$



$$\mathcal{E} = \{x \in \mathbb{N} : x \leq 12\}$$

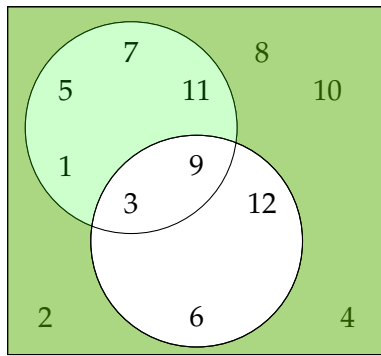
$$A = \{x : x \text{ er oddetall}\}$$

$$B = \{x : x > 7\}$$

$$C = \{x : x \text{ er delelig med } 3\}$$

Svar (d): {3, 6, 9}

(e) $\overline{(A \cup C)} \cup \bar{C}$



$$E = \{x \in \mathbb{N} : x \leq 12\}$$

$$A = \{x : x \text{ er oddetall}\}$$

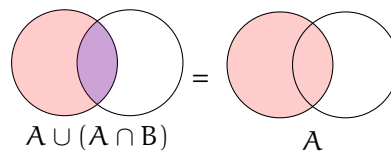
$$B = \{x : x > 7\}$$

$$C = \{x : x \text{ er delelig med } 3\}$$

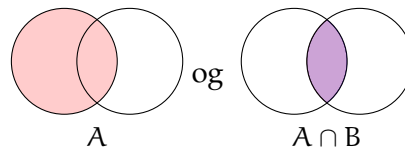
Svar (e): $\{1, 2, 4, 5, 7, 8, 10, 11\}$

Oppgave 5.6

Illustrer den andre absorpsjonsloven, $A \cup (A \cap B) = A$, ved å bruke Venn-diagrammer.

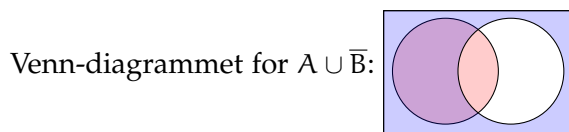
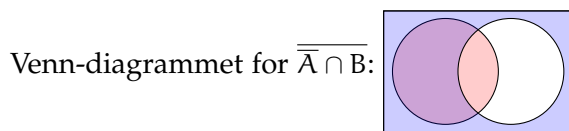


Diagrammet for $A \cup (A \cap B)$ får vi ved å slå sammen følgende diagrammer:



Oppgave 5.7 (med Venn-diagrammer)

Vis at $\overline{\bar{A} \cap B} = A \cup \bar{B}$ ved å bruke Venn-diagrammer.



Siden Venn-diagrammene er like, må $\overline{\bar{A} \cap B} = A \cup \bar{B}$.

Oppgave 5.8

Er påstanden "enhver mengde med n elementer har 2^n delmengder" sann når $n = 0$?

Løsning

- Når $n = 0$ er mengden tom.
- Hvor mange delmengder har en tom mengde?
- Den har nøyaktig én delmengde, nemlig den tomme mengden.
- Siden $2^0 = 1$, blir svaret ja.

Oppgave 5.9

La $A = \{a, b, c\}$ og $B = \{p, q\}$. Skriv ned følgende mengder på listeform.

- (a) $A \times B$
- (b) A^2
- (c) B^3

Løsning

- (a) $A \times B = \{(a, p), (a, q), (b, p), (b, q), (c, p), (c, q)\}$
- (b) $A^2 = \{(a, a), (a, b), (a, c), (b, a), (b, b), (b, c), (c, a), (c, b), (c, c)\}$
- (c) $B^3 = \{(p, p, p), (p, p, q), (p, q, p), (p, q, q), (q, p, p), (q, p, q), (q, q, p), (q, q, q)\}$

Oppgave 5.10

Uttrykk hver av følgende mengder som et Kartesisk produkt av mengder.

Løsning

Eventuelt på tavla.

Oppgave 5.13

Se på følgende algoritme.

1. Input en bitstreng b
2. $n \leftarrow$ det usignerte bitet som har b som sin binære representasjon
3. teller $\leftarrow 0$
4. **While** $n > 0$ **do**
 - 4.1. $n \leftarrow n \wedge (n-1)$ [\wedge betyr her bitvis 'og', her anvendt på den binære representasjonen av n og $n-1$]
 - 4.2. teller \leftarrow teller + 1 [skrivefeil i boka: det står $-$ i stedet for $+$]
5. Output teller

Vis at returverdien er antall enere i b .

Eksempel.

Anta at input er 100101.

teller	binær representasjon av n og $n-1$
0	1001001 1001000
1	1001000 1000111
2	1000000 0111111
3	0000000

Eksempel.

Anta at input er 111101.

teller	binær representasjon av n og $n - 1$
0	1111001 1111000
1	1111000 1110111
2	1110000 1101111
3	1100000 1011111
4	1000000 0111111
5	0000000

Løsning

- Vi skal vise at returverdien er antall enere i inputverdien.
- Det er tilstrekkelig å vise at hvert steg i **While**-løkken reduserer antall enere med nøyaktig én.
- Vi må vise at bitvis \wedge av representasjonene til n og $n - 1$ reduserer antall enere med nøyaktig én.
- La $b_1 \dots b_k$ være representasjonen til n .
- Vi kan anta at $n > 0$. Da må minst et bit være en ener.
- La b_m være eneren som forekommer lengst til høyre, for $1 \leq m \leq k$.
- Vi har følgende situasjon, hvor vi ser at bitvis \wedge fjerner nøyaktig én ener. (Eventuelt flere detaljer i plenum.)

$$\begin{array}{r} b_1 b_2 \dots b_{m-1} 100 \dots 00 \\ b_1 b_2 \dots b_{m-1} 011 \dots 11 \\ \hline b_1 b_2 \dots b_{m-1} 000 \dots 00 \end{array}$$

Oppgave 5.14

Se på følgende sekvens av steg i pseudokode, hvor x og y er bitstrenger av lik lengde.

1. $x \leftarrow x \oplus y$ [\oplus står for bitvis eksklusiv eller]

2. $y \leftarrow x \oplus y$

3. $x \leftarrow x \oplus y$

Vis at effekten av sekvensen av steg er å bytte verdiene til x og y .

Løsning

- La $x = x_1 \dots x_n$ og $y = y_1 \dots y_n$.
- La oss se på hva som skjer bitvis, for x_i og y_i , hvor $1 \leq i \leq n$.
- Steg 1, $x \leftarrow x \oplus y$, setter x_i lik $x_i \oplus y_i$.
- Steg 2, $y \leftarrow x \oplus y$, setter y_i lik $(x_i \oplus y_i) \oplus y_i$.
- Steg 3, $x \leftarrow x \oplus y$, setter x_i lik $(x_i \oplus y_i) \oplus ((x_i \oplus y_i) \oplus y_i)$.
- Vi må vise at $(x_i \oplus y_i) \oplus y_i$ er lik den opprinnelige x_i .
- Vi må vise at $(x_i \oplus y_i) \oplus ((x_i \oplus y_i) \oplus y_i)$ er lik den opprinnelige y_i .

x_i	y_i	$(x_i \oplus y_i)$	\oplus	$((x_i \oplus y_i) \oplus y_i)$	\oplus	y_i
1	1	0	1	0	1	1
1	0	1	0	1	1	0
0	1	1	1	1	0	1
0	0	0	0	0	0	0

- Kolonnene 1 og 6 er like, og kolonnene 2 og 4 er like.

Løsning

- Vi kunne også ha løst oppgaven ved å vise at $(p \oplus q) \oplus q$ er logisk ekvivalent med p .

p	q	$(p \oplus q)$	\oplus	q
1	1	0	1	1
1	0	1	1	0
0	1	1	0	1
0	0	0	0	0

- Da må $(x_i \oplus y_i) \oplus y_i$ (den nye verdien til y_i) være lik x_i
- Da må $(x_i \oplus y_i) \oplus ((x_i \oplus y_i) \oplus y_i)$ (den nye verdien til x_i) være lik y_i .

Oppgave 5.15

La $A = \{1, 2, 3, 4, 5\}$ og la R være relasjonen på A som er definert slik:

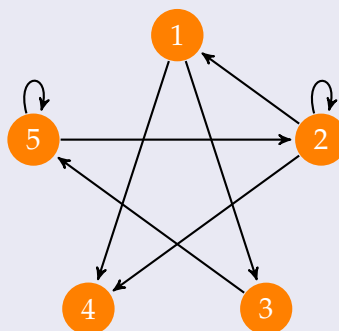
$$R = \{(1, 3), (1, 4), (2, 1), (2, 2), (2, 4), (3, 5), (5, 2), (5, 5)\}$$

- Skriv ned relasjonen R på matriseform.
- Tegn den grafiske representasjonen av R .

Løsning

Vi gir relasjonen på matriseform og tegner den grafiske representasjonen.

$$\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{bmatrix} & 1 & 2 & 3 & 4 & 5 \\ \text{F} & \text{F} & \text{T} & \text{T} & \text{F} \\ \text{T} & \text{T} & \text{F} & \text{T} & \text{F} \\ \text{F} & \text{F} & \text{F} & \text{F} & \text{T} \\ \text{F} & \text{F} & \text{F} & \text{F} & \text{F} \\ \text{F} & \text{T} & \text{F} & \text{F} & \text{T} \end{bmatrix}$$



Oppgave fra forelesning 04.02.2008, side 9

- Vis at hvis A , B og C er sammensatte utsagn, så vil

$$(A \leftrightarrow B) \leftrightarrow C \equiv A \leftrightarrow (B \leftrightarrow C)$$

og at

$$(A \leftrightarrow B) \equiv (B \leftrightarrow A).$$

- Forklar hvorfor dette betyr at rekkefølge og parentessetting ikke betyr noe i et utsagnslogisk uttrykk som bare bruker bindeordet \leftrightarrow
- [Vanskelig] Hvordan kan vi lett avgjøre om et slikt uttrykk er en tautologi eller ikke?

Løsning (a)

A	B	C	$(A \leftrightarrow B)$	\leftrightarrow	C	A	\leftrightarrow	$(B \leftrightarrow C)$
T	T	T	T	T	T	T	T	T
T	T	F	T	F	F	T	F	F
T	F	T	F	F	T	T	F	F
T	F	F	F	T	F	T	T	T
F	T	T	F	F	T	F	F	T
F	T	F	F	T	F	F	T	F
F	F	T	T	T	T	F	T	F
F	F	F	T	F	F	F	F	T

A	B	$(A \leftrightarrow B)$	$(B \leftrightarrow A)$
T	T	T	T
T	F	F	F
F	T	F	F
F	F	T	T

Løsning (b)

Den første sannhetsverditabellen fra (a) gir oss at parentessetting er overflødig når vi kun bruker \leftrightarrow . Vi kan skrive $A \leftrightarrow B \leftrightarrow C$ i stedet for både $A \leftrightarrow (B \leftrightarrow C)$ og $(A \leftrightarrow B) \leftrightarrow C$. Eksempel:

- $(A \leftrightarrow B) \leftrightarrow (C \leftrightarrow (D \leftrightarrow E)) \equiv$
- $(A \leftrightarrow B) \leftrightarrow (C \leftrightarrow D \leftrightarrow E) \equiv$
- $A \leftrightarrow B \leftrightarrow C \leftrightarrow D \leftrightarrow E$

Den andre sannhetsverditabellen fra (a) gir oss at rekkefølgen ikke spiller noen rolle. F.eks. kan vi skrive $A \leftrightarrow B$ i stedet for $B \leftrightarrow A$.

Løsning (c)

Siden hverken rekkefølge eller parentessetting spiller noen rolle for sannhetsverdien til et uttrykk som består av \leftrightarrow , så står vi fritt til å gruppere utsagnsvariablene som vi ønsker. Det kan da være en idé å sette like utsagnsvariable ved siden av hverandre. Eksempel:

- $((A \leftrightarrow B) \leftrightarrow (C \leftrightarrow B)) \leftrightarrow A$
- Ved å ta bort parentesene får vi $A \leftrightarrow B \leftrightarrow C \leftrightarrow B \leftrightarrow A$.
- Ved å endre på rekkefølgen får vi $A \leftrightarrow A \leftrightarrow B \leftrightarrow B \leftrightarrow C$.

- Ved å sette på parenteser igjen får vi $(A \leftrightarrow A) \leftrightarrow ((B \leftrightarrow B) \leftrightarrow C)$.
- Vi ser at $(A \leftrightarrow A)$ og $(B \leftrightarrow B)$ er tautologier, så uttrykket er ekvivalent med C , som ikke er en tautologi.

Hvis uttrykket hadde bestått av et partall forekomster av hver utsagnsvariabel, så ville to og to like utsagnsvariable kansellert hverandre, helt til vi hadde hatt kun en tautologi til slutt.

Løsning (c)

Et utsagnslogisk uttrykk som kun inneholder bindeordet \leftrightarrow er en tautologi hvis og bare hvis det inneholder et partall forekomster av hver utsagnsvariabel.

Plenumsregning 7

Ukeoppgaver fra kapittel 5 & 6, mm.

Roger Antonsen - 28. februar 2008

Oppgave 5.16

La R være relasjonen på $\{a, b, c, d\}$ definert av følgende matrise.

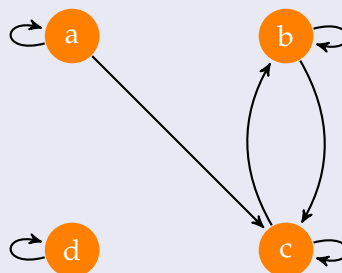
$$\begin{array}{c} 1 \quad 2 \quad 3 \quad 4 \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{bmatrix} \mathbf{T} & \mathbf{F} & \mathbf{T} & \mathbf{F} \\ \mathbf{F} & \mathbf{T} & \mathbf{T} & \mathbf{F} \\ \mathbf{F} & \mathbf{T} & \mathbf{T} & \mathbf{F} \\ \mathbf{F} & \mathbf{F} & \mathbf{F} & \mathbf{T} \end{bmatrix} \end{array}$$

- (a) Tegn den grafiske representasjonen av R .
(b) Finn ut, og gi grunner for hvorvidt, R er refleksiv, symmetrisk eller transitiv.

Løsning

Vi tegner den grafiske representasjonen.

$$\begin{array}{c} a \quad b \quad c \quad d \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} \begin{bmatrix} \mathbf{T} & \mathbf{F} & \mathbf{T} & \mathbf{F} \\ \mathbf{F} & \mathbf{T} & \mathbf{T} & \mathbf{F} \\ \mathbf{F} & \mathbf{T} & \mathbf{T} & \mathbf{F} \\ \mathbf{F} & \mathbf{F} & \mathbf{F} & \mathbf{T} \end{bmatrix} \end{array}$$



- Er R refleksiv? **Ja** Hvorfor? Fordi alle på diagonalen er **T**. Og fordi alle nodene har en pil til seg selv.
- Er R symmetrisk? **Nei** Hvorfor ikke? Fordi matrisen ikke kan speiles om diagonalen. Vi har $(a, c) \in R$, men *ikke* $(c, a) \in R$.
- Er R transitiv? **Nei** Hvorfor ikke? Vi har $(a, c) \in R$ og $(c, b) \in R$, men *ikke* $(a, b) \in R$.

Oppgave 5.17

Er matriserepresentasjonen av en relasjon unik eller kan den samme relasjonen representeres av to forskjellige matriser?

Løsning

Representasjonen er ikke unik, siden den samme relasjonen kan representeres av to forskjellige matriser.

$$\begin{array}{cc} & \begin{array}{cc} a & b \end{array} \\ \begin{array}{c} a \\ b \end{array} & \begin{bmatrix} F & T \\ T & T \end{bmatrix} \end{array} \qquad \begin{array}{cc} & \begin{array}{cc} b & a \end{array} \\ \begin{array}{c} b \\ a \end{array} & \begin{bmatrix} T & T \\ T & F \end{bmatrix} \end{array}$$

Begge matrisene representerer relasjonen $\{(a, b), (b, a), (b, b)\}$.

Oppgave 5.18

Avgjør om følgende relasjoner refleksive, irrefleksive, symmetriske, antisymmetriske eller transitive.

- (a) "er søsken (bror eller søster) til", på mengden av alle mennesker
- (b) "er sønnen til", på mengden av alle mennesker
- (c) "er større enn", på mengden av reelle tall
- (d) relasjonen R på reelle tall definert ved xRy hvis $x^2 = y^2$
- (e) "har samme heltallsdel som", på mengden av reelle tall
- (f) "er et multiplum av", på mengden av positive heltall

Oppgave 5.19

Avgjør hvilke av relasjonene i Oppgave 5.18 som er ekvivalensrelasjoner. For de som er ekvivalensrelasjoner, beskriv ekvivalensklassene.

Oppgave 5.20

Avgjør hvilke av relasjonene i Oppgave 5.18 som er partielle ordninger.

Løsning

(a) "er søsken (bror eller søster) til", på mengden av alle mennesker

- Refleksiv? **Nei**, det fins en som ikke er sin egen bror eller søster.
- Irrefleksiv? **Ja**, ingen er sin egen bror eller søster.
- Symmetrisk? **Ja**, hvis x er søsken til y , så er y søsken til x .
- Antisymmetrisk? **Nei**, vi ha at x er søsken til y og at y søsken til x , uten at $x = y$.
- Transitiv? **Nei**, hvis vi tillater halvsøsken, så kan a være søsken til b , og b søsken til c , uten at a er søsken til c . En annen grunn er at a kan være søsken til b og b søsken til a , men a kan er ikke søsken til a . (Takk til oppmerksomme studenter!)
- Ekvivalensrelasjon? **Nei**, den er ikke refleksiv
- Partiell ordning? **Nei**, den er hverken refleksiv, antisymmetrisk eller transitiv.

Løsning

(b) "er sønnen til", på mengden av alle mennesker

- Refleksiv? **Nei**, det fins en som ikke er sin egen sønn.
- Irrefleksiv? **Ja**, ingen er sin egen sønn.
- Symmetrisk? **Nei**, fordi "X er sønnen til Y" ikke medfører at "Y er sønnen til X".
- Antisymmetrisk? **Ja**, fordi "X er sønnen til Y" og "Y er sønnen til X" aldri er sanne samtidig.
- Transitiv? **Nei**, "X er sønnen til Y" og "Y er sønnen til Z" ikke medfører at "X er sønnen til Z".
- Ekvivalensrelasjon? **Nei**, den er hverken refleksiv, symmetrisk eller transitiv.
- Partiell ordning? **Nei**, den er hverken refleksiv eller transitiv.

Løsning

(c) "er større enn", på mengden av reelle tall

- Refleksiv? **Nei**, det fins et reellt tall som ikke er større enn seg selv.
- Irrefleksiv? **Ja**, ingen reelle tall er større enn seg selv.
- Symmetrisk? **Nei**, f.eks. er 3 større enn 2, men 2 er ikke større enn 3.
- Antisymmetrisk? **Ja**, vi kan ikke ha at to tall er større enn hverandre.
- Transitiv? **Ja**, hvis x er større enn y og y er større enn z , så er x større enn z .
- Ekvivalensrelasjon? **Nei**, den er hverken refleksiv eller symmetrisk.
- Partiell ordning? **Nei**, den er ikke refleksiv.

Løsning

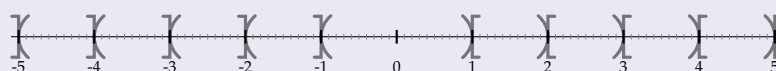
(d) relasjonen R på reelle tall definert ved xRy hvis $x^2 = y^2$

- Refleksiv? **Ja**, siden $x^2 = x^2$.
- Irrefleksiv? **Nei**, nei, f.eks. har vi at $1R1$.
- Symmetrisk? **Ja**, hvis $x^2 = y^2$ holder, så vil også $y^2 = x^2$ holde.
- Antisymmetrisk? **Nei**, vi har at $1R(-1)$ og $(-1)R1$, men $1 \neq -1$.
- Transitiv? **Ja**, hvis $x^2 = y^2$ og $y^2 = z^2$, så $x^2 = z^2$.
- Ekvivalensrelasjon? **Ja**
- Ekvivalensklasser: Alle mengder $\{x, -x\}$, hvor x er et positivt reelt tall, samt $\{0\}$.
- Partiell ordning? **Nei**, den er ikke antisymmetrisk.

Løsning

(e) "har samme heltallsdel som", på mengden av reelle tall

- Refleksiv? **Ja**, ethvert tall har samme heltallsdel som seg selv.
- Irrefleksiv? **Nei**, f.eks. har 3, 14 samme heltallsdel som seg selv.
- Symmetrisk? **Ja**, hvis x har samme heltallsdel som y , så må y ha samme heltallsdel som x .
- Antisymmetrisk? **Nei**, vi har at 1, 28 og 1, 32 har samme heltallsdel, men de er ikke like.
- Transitiv? **Ja**
- Ekvivalensrelasjon? **Ja**
- Ekvivalensklasser: For hvert heltall n har vi en ekvivalensklasse som består av de reelle tall med heltallsdel lik n :



Ekvivalensklassene er følgende mengder, for $n \in \mathbb{N}$: $\{x : -n - 1 < x \leq -n\}$ og $\{x : -1 < x < 1\}$ og $\{x : n \leq x < n + 1\}$

(Fasiten i boka sier $\{x : -n < x \leq -n + 1\}$, men det blir feil for $n = 1$.)

- Partiell ordning? **Nei**, den er ikke antisymmetrisk.

Løsning

(f) "er et multiplum av", på mengden av positive heltall

- Refleksiv? **Ja**, ethvert tall er et multiplum av seg selv.
- Irrefleksiv? **Nei**, f.eks. er 2 et multiplum av seg selv.

- Symmetrisk? **Nei**, 4 er et multippel av 2, og 2 er ikke et multippel av 4.
- Antisymmetrisk? **Ja**
 - Anta at $x = ay$ og at $y = bx$, for positive heltall x, y, a, b .
 - Da må $x = abx$.
 - Siden alle tallene er positive heltall, må $ab = 1$, og dermed må $a = 1$ og $b = 1$.
 - Siden $x = ay$, må $x = y$.
- Transitiv? **Ja**
 - Anta at $x = ay$ og at $y = bz$, for positive heltall x, y, z, a, b .
 - Da må $x = abz$, og x er et multippel av z .
- Ekvivalensrelasjon? **Nei**, den er ikke symmetrisk.
- Partiell ordning? **Ja**

Oppgave 5.21

Et dataprogram består av fem moduler: M_1, M_2, \dots, M_5 . En relasjon R på mengden av moduler er definert ved at $M_i R M_j$ hvis M_i er i kallsekvensen til M_j . Relasjonsmatrisen til R er denne:

$$\begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} \mathbf{T} & \mathbf{F} & \mathbf{T} & \mathbf{T} & \mathbf{F} \\ \mathbf{F} & \mathbf{T} & \mathbf{T} & \mathbf{F} & \mathbf{F} \\ \mathbf{F} & \mathbf{F} & \mathbf{T} & \mathbf{F} & \mathbf{F} \\ \mathbf{F} & \mathbf{F} & \mathbf{T} & \mathbf{T} & \mathbf{F} \\ \mathbf{F} & \mathbf{F} & \mathbf{T} & \mathbf{T} & \mathbf{T} \end{bmatrix} \end{matrix}$$

- (a) Bekreft at R er refleksiv, antisymmetrisk og transitiv.
(b) Hvilken modul er hovedprogrammet?

Løsning

$$\begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} \mathbf{T} & \mathbf{F} & \mathbf{T} & \mathbf{T} & \mathbf{F} \\ \mathbf{F} & \mathbf{T} & \mathbf{T} & \mathbf{F} & \mathbf{F} \\ \mathbf{F} & \mathbf{F} & \mathbf{T} & \mathbf{F} & \mathbf{F} \\ \mathbf{F} & \mathbf{F} & \mathbf{T} & \mathbf{T} & \mathbf{F} \\ \mathbf{F} & \mathbf{F} & \mathbf{T} & \mathbf{T} & \mathbf{T} \end{bmatrix} \end{matrix}$$

- R er refleksiv: Alle verdier på diagonalen er \mathbf{T} .
- R er antisymmetrisk: Fordi ingen \mathbf{T} speiles om diagonalen.
- R er transitiv: Vi må sjekke alle muligheter i matrisen.
- Hvilken modul er hovedprogrammet? Oppgaven sa at $M_i R M_j$ holder hvis M_i er i kallsekvensen til M_j . M_3 skiller seg ut. Den kaller på alle moduler og blir kun kalt på av seg selv. Dette må være hovedprogrammet.

Oppgave 6.1

Avgjør hvorvidt følgende funksjoner er veldefinerte. For de som er veldefinerte, gi definisjonsområdet, verdiområdet og bildemengden.

[Se læreboken på side 107.]

Løsning

Vi setter opp en tabell.

	Veldefinert?	Definisjonsområde	Verdiområde	Bildemengde
(a)	Ja	\mathbb{R}	\mathbb{R}	\mathbb{R}
(b)	Nei, $g(1)$ er ikke i \mathbb{J} .			
(c)	Nei, $h(3) = 4$ er ikke i $\{1, 2, 3\}$.			
(d)	Ja	\mathbb{N}	\mathbb{N}	\mathbb{N}
(e)	Ja	$\mathbb{R} - \{0\}$	\mathbb{R}	$\mathbb{R} - \{0\}$
(f)	Nei, $i_{\text{positive}}(0)$ er ikke definert.			
(g)	Ja	\mathbb{N}	\mathbb{J}	$\{0, 1, 2, 3, 4, 5\}$
(h)	Nei, $\psi(a)$ er den tomme strengen og er ikke med i S .			

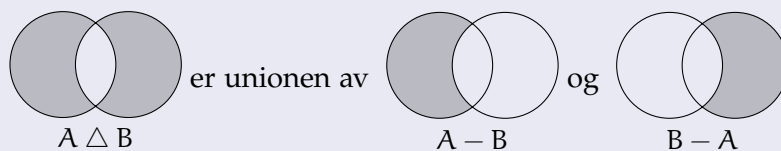
Oppgave på side 6, fra forelesningen 13/2

Vi definerer ofte symmetrisk differens ved

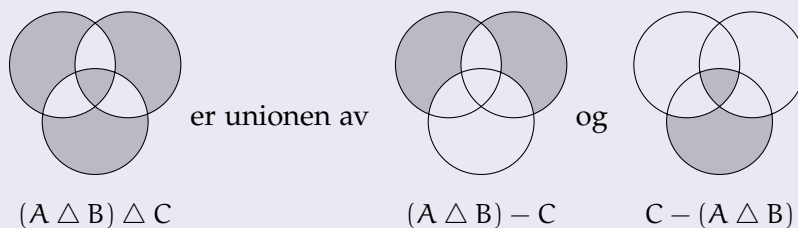
$$A \triangle B = (A - B) \cup (B - A).$$

- Illustrer $A \triangle B$ ved et Venn-diagram.
- Vis at $(A \triangle B) \triangle C$ kan illustreres ved Venn-diagrammet på neste side.
- Drøft hvorfor dette viser at vi kunne skrevet $A \triangle B \triangle C$ uten bruk av parenteser.

Løsning (a)



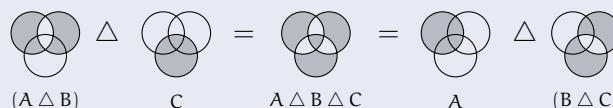
Løsning (b)



Symmetrisk differanse inverterer fargene.

Løsning (c)

- Vi har vist at $(A \Delta B) \Delta C$ kan tegnes med diagrammet
- Det er lett å se at $A \Delta (B \Delta C)$ også har diagrammet
- Vi kan illustrere det slik:

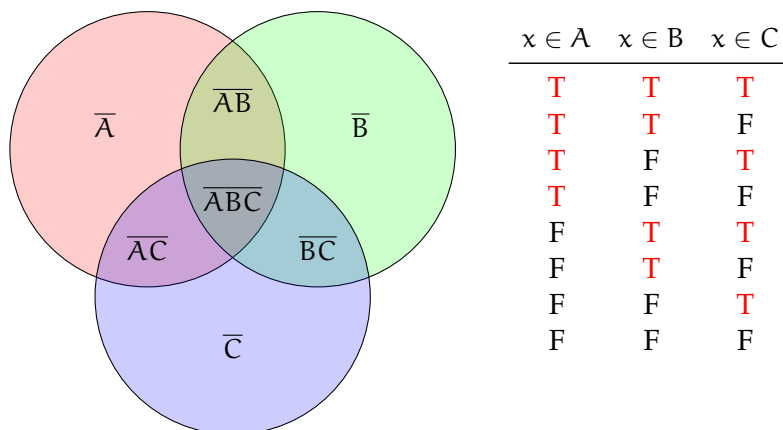


- Parentessettingen har ingenting å si for hvordan Venn-diagrammet ser ut.

Oppgave på side 8, fra forelesningen 13/2

- Vi bruker bare Venn-diagrammer for uttrykk med en, to eller tre mengder.
- Tegn et Venn-diagram for tre mengder A , B og C , og sett inn sannhetsverdiene for de tre basisutsagnene $x \in A$, $x \in B$ og $x \in C$ i de forskjellige feltene.

Vi skriver sannhetsverdiene for $x \in A$, $x \in B$ og $x \in C$ som henholdsvis \bar{A} , \bar{B} og \bar{C} .

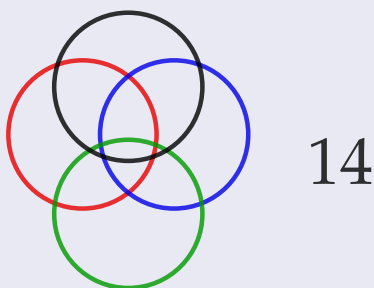


Oppgave på side 8, fra forelesningen 13/2

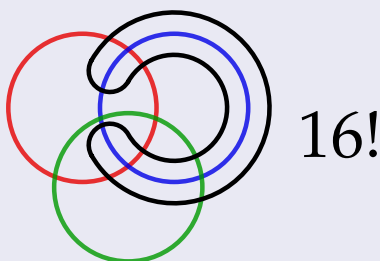
- Undersøk hvor mange deler det er mulig å dele planet inn i ved hjelp av fire sirkler.
- Forklar hvorfor dette viser at Venn-diagrammer ikke er hensiktsmessige for Booleske uttrykk med mer enn tre mengder.

Løsning

- Med tre sirker får vi åtte områder, som svarer til de åtte mulige kombinasjonene.
- Hva med fire sirkler? Hvor mange områder kan vi dele planet opp i med fire sirkler?



- Hvis ellipser eller andre kurver er tillatt, klarer vi å lage Venn-diagrammer for fire (eller flere) mengder.



Oppgave på side 16, fra forelesningen 20/2

La A være en mengde og la R og S være ekvivalensrelasjoner på A .

La $T = S \cap R$

- Forklar hvorfor vi har at aTb hvis og bare hvis $aRb \wedge aSb$ for alle a og b i A .
- Vis at T er en ekvivalensrelasjon.
- Finn et eksempel hvor $R \cup S$ ikke er en ekvivalensrelasjon.

Løsning

Denne løser vi i plenum på tavla.

Oppgave på side 23/24, fra forelesningen 20/2 (for spesielt interesserte)

La R være en relasjon på en mengde A .

Vi kaller R en preordning hvis R er transitiv og reflektiv.

Definer relasjonen S på A ved aSb når $aRb \wedge bRa$.

a) Vis at S er en ekvivalensrelasjon på A .

Der er vanlig å skrive A/S for mengden av ekvivalensklasser $E(a)$ til ekvivalensrelasjonen S .

Løsning (a)

- S er reflektiv:
 - La a være et vilkårlig element fra A .
 - Siden R er reflektiv, må aRa .
 - Da holder $aRa \wedge aRa$.
 - Da holder aSa .
- S er symmetrisk:
 - Anta at aSb , for vilkårlige elementer a og b fra A .
 - Da holder $aRb \wedge bRa$ per definisjon av S .
 - Da må bSa holde.
- S er transitiv:
 - Anta aSb og bSc , for vilkårlige elementer a , b og c fra A .
 - Per definisjon av S har vi $aRb \wedge bRa$ og $bRc \wedge cRb$.
 - Siden aRb og bRc , og R er transitiv, vil aRc .
 - Siden cRb og bRa , og R er transitiv, vil cRa .
 - Da holder $aRc \wedge cRa$, og dermed aSc .

Oppgave på side 23/24, fra forelesningen 20/2 (for spesielt interesserte)

Vi definerer en relasjon \hat{R} på A/S ved

$$E(a)\hat{R}E(b)$$

om aRb

- b) Vis at det ikke spiller noen rolle hvilke elementer i ekvivalensklassene $E(a)$ og $E(b)$ vi bruker.
- c) Vis at \hat{R} er en partiell ordning på mengden av ekvivalensklasser.

Løsning (b)

- Anta at aRb , og at $a' \in E(a)$ og $b' \in E(b)$.
- Det er tilstrekkelig å vise at $a'Rb'$, for det betyr at det ikke spiller noen rolle hvilke elementer i ekvivalensklassene vi bruker når vi definerer \hat{R} .
- Siden a og a' er i samme ekvivalensklasse, må aSa' , det vil si aRa' og $a'Ra$.
- Siden b og b' er i samme ekvivalensklasse, må bSb' , det vil si bRb' og $b'Rb$.
- $a'Ra$ og aRb gir at $a'Rb$, siden R er transitiv.
- $a'Rb$ og bRb' gir at $a'Rb'$, siden R er transitiv.
- Tegn figur!

Løsning (c)

- \hat{R} er refleksiv, siden $E(a)\hat{R}E(a)$ holder for alle $E(a)$.
- \hat{R} er transitiv:
 - Anta at $E(a)\hat{R}E(b)$ og $E(b)\hat{R}E(c)$.
 - Da må aRb og bRc .
 - Siden R er transitiv, må aRc .
 - Da må $E(a)\hat{R}E(c)$.
- \hat{R} er antisymmetrisk.
 - Anta at $E(a)\hat{R}E(b)$ og $E(b)\hat{R}E(a)$.
 - Da må aRb og bRa .
 - Da må aSb .
 - Da må $E(a)$ og $E(b)$ være samme ekvivalensklasse.

Plenumsregning 8

Diverse ukeoppgaver fra kapittel 6 & 7, mm.

Roger Antonsen - 27. mars 2008

Repetisjon

- Funksjoner - Viktige begreper
 - Argumenter og verdier
 - Definisjonsområdet til en funksjon
 - Verdiområdet til en funksjon
 - Bildet til en funksjon
 - Injektive funksjoner
 - Surjektive funksjoner
 - Sammensetning av funksjoner
 - Inverse funksjoner

Repetisjon

- Rekursjon og induksjon
 - Rekursive definisjoner
 - Induksjonsbevis
 - Induksjonsstart & induksjonsskritt
 - Rekurrenslikninger
 - Hvordan finne løsninger til rekurrenslikninger
 - Den karakteristiske likningen til en rekurrenslikning
 - Generell rekursjon og induksjon
 - Induktivt definerte mengder
 - Rekursjon over slike
 - Alfabet, ord, formelle språk, formler
 - Simultan rekursjon

Oppgave (fra forelesningen 25/2 på side 33).

- a) Vis at hvis $f : X \rightarrow Y$ og $g : Y \rightarrow Z$ begge har inverse f^{-1} og g^{-1} , så vil sammensetningen

$$h = g \circ f$$

også ha en invers.

- b) Under antagelsene i a), vis at $h^{-1} = f^{-1} \circ g^{-1}$.

Løsning (a)

- Det går fint an å vise dette direkte, men vi kan også bruke det vi har lært om funksjoner.
- (1) Vi har tidligere i kurset vist at en funksjon har en invers hvis og bare hvis den er injektiv og surjektiv. Det er derfor tilstrekkelig å vise at h er både injektiv og surjektiv.
 - Ved (1), siden f og g begge har inverser, så må f og g være både injektive og surjektive.
 - (2) Vi har tidligere vist at hvis f og g er injektive, så er $g \circ f$ injektiv.
 - (3) Vi har tidligere vist at hvis f og g er surjektive, så er $g \circ f$ surjektiv.
 - Ved (2) og (3) er h både injektiv og surjektiv.
 - Ved (1) har h en invers.

Løsning (b)

- For å vise at $f^{-1} \circ g^{-1}$ er inversen til h , må vi vise at $h((f^{-1} \circ g^{-1})(z)) = z$ for alle $z \in Z$ og at $(f^{-1} \circ g^{-1})(h(x)) = x$ for alle $x \in X$.
- For å gjøre det må vi bruke at f^{-1} er inversen til f og at g^{-1} er inversen til g .
- $h((f^{-1} \circ g^{-1})(z)) = h(f^{-1}(g^{-1}(z))) = (g \circ f)(f^{-1}(g^{-1}(z))) = g(f(f^{-1}(g^{-1}(z)))) = g(g^{-1}(z)) = z$
- $(f^{-1} \circ g^{-1})(h(x)) = f^{-1}(g^{-1}(h(x))) = f^{-1}(g^{-1}((g \circ f)(x))) = f^{-1}(g^{-1}(g(f(x)))) = f^{-1}(f(x)) = x$

Oppgave (fra forelesningen 3/3 på side 20).

- Vi vet at vi kan dele planet opp i to felter ved hjelp av en sirkel.
- Vi vet at to sirkler kan skjære hverandre i to punkter
- Vi vet at $2n$ punkter vil dele en sirkel opp i $2n$ buestykker.
- Bruk dette til å definere funksjonen $G(n)$ ved rekursjon, hvor $G(n)$ er antall områder vi kan dele planet opp i ved hjelp av n sirkler.
- Foreslå en formel for $G(n)$ og se om du kan vise den ved induksjon.

Løsning

- $G(1) = 2$, siden én sirkel gir to områder.
- Hvis n sirkler deler planet opp i $G(n)$ områder, må vi se på hva som skjer når vi legger til en ny sirkel.
- Siden den nye sirkelen vil skjære hver av de andre n sirklene høyst 2 steder, så får vi høyst $2n$ nye skjæringspunkter.

- Disse skjæringspunktene deler den nye sirkelen inn i $2n$ buestykker, og hvert av disse buestykkene kan dele et av de gamle områdene inn i to.
- Vi kan altså få $2n$ nye områder når vi legger til sirkel nummer $n + 1$.
- Det betyr at $G(n + 1) = G(n) + 2n$

Løsning

- Hvis vi skal tippe på en formel direkte, så ser vi at $G(n)$ ligger ganske nærme n^2 .

n	$G(n)$	n^2	$G(n) - n^2$
1	2	1	1
2	4	4	0
3	8	9	-1
4	14	16	-2
5	22	25	-3
6	32	36	-4

- Vi ser at differansen mellom $G(n)$ og n^2 er $-n + 2$.
- Vi tipper derfor på formelen $G(n) = n^2 - n + 2$.
- Vi viser at $G(n) = n^2 - n + 2$ ved induksjon.

Løsning

- Basissteget er å vise at formelen $G(n) = n^2 - n + 2$ holder for $n = 1$: $G(1) = 1^2 - 1 + 2 = 2$, som stemmer.
- Vi antar så at $G(n) = n^2 - n + 2$. Det er *induksjonshypotesen*.
- *Induksjonssteget* går ut på å vise at $G(n + 1) = (n + 1)^2 - (n + 1) + 2$.
- Vi regner på følgende måte:

$$\begin{aligned}
 G(n + 1) &= G(n) + 2n && \text{(ved antakelse)} \\
 &= (n^2 - n + 2) + 2n && \text{(ved induksjonshypotese)} \\
 &= n^2 + n + 2 \\
 &= (n^2 + 2n + 1) - n + 1 \\
 &= (n + 1)^2 - n + 1 \\
 &= (n + 1)^2 - (n + 1) + 2
 \end{aligned}$$

Oppgave 7.6

Finn en rekursiv og en ikke-rekursiv definisjon av sekvensen

$$2, 7, 12, 17, 22, 27, \dots$$

Løsning

- En rekursiv definisjon er slik:
 - $f(1) = 2$
 - $f(n + 1) = f(n) + 5$
- En ikke-rekursiv definisjon er slik:
 - $g(n) = 5n - 3$

Oppgave 7.12 (b)

Vis ved induksjon at følgende påstand er sann for alle naturlige tall n .

$$1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Løsning

Vi viser først at påstanden holder for $n = 1$.

$$1^2 = \frac{1(1+1)(2 \cdot 1 + 1)}{6} = \frac{6}{6} = 1$$

Løsning

Vi antar så at påstanden holder for $n = k$ (det er vår induksjonshypotese) og viser at den

holder for $n = k + 1$.

$$\begin{aligned} & 1^2 + 2^2 + 3^2 + \dots + k^2 + (k + 1)^2 \\ &= \frac{k(k + 1)(2k + 1)}{6} + (k + 1)^2 \quad (\text{ved induksjonshypotesen}) \\ &= \frac{k(k + 1)(2k + 1) + 6(k + 1)(k + 1)}{6} \\ &= \frac{(k + 1)(k(2k + 1) + 6(k + 1))}{6} \\ &= \frac{(k + 1)(2k^2 + 7k + 6)}{6} \\ &= \frac{(k + 1)(k + 2)(2k + 3)}{6} = \frac{(k + 1)(k + 2)(2(k + 1) + 1)}{6} \end{aligned}$$

Oppgave 7.12 (c)

Vis ved induksjon at følgende påstand er sann for alle naturlige tall n .

$$n^3 - n \text{ er delelig med } 3$$

Løsning

Vi viser først at påstanden holder for $n = 1$.

$$1^3 - 1 = 0 \text{ er delelig med } 3$$

Løsning

- Vi antar så at påstanden holder for $n = k$.
- Det er *induksjonshypotesen*.
- Vi viser så at påstanden holder for $n = k + 1$.
- Induksjonshypotesen sier i dette tilfellet at $k^3 - k$ er delelig med 3.
- Vi kan derfor anta at det fins et naturlig tall p slik at $k^3 - k = 3p$.

$$\begin{aligned} (k + 1)^3 - (k + 1) &= (k^3 + 3k^2 + 3k + 1) - (k + 1) \\ &= k^3 + 3k^2 + 2k \\ &= (k^3 - k) + 3k^2 + 3k \\ &= 3p + 3(k^2 + k) \quad (\text{ved induksjonshypotesen}) \\ &= 3(p + k^2 + k) \end{aligned}$$

- Da må $(k + 1)^3 - (k + 1)$ være delelig på 3 og påstanden holder.

Oppgave 7.20

(Om å gjøre induksjonsbevis ved å starte med $n = 0$.)

- Bevis at $(6^n - 1)/5$ er et heltall for $n \geq 0$, ved å bruke $n = 0$ som induksjonsstart.
- Forklar problemet ved å vise at $2^n > n$ for $n \geq 0$ ved induksjon hvor $n = 0$ er induksjonsstarten.
- Bruk induksjon på en annen måte for å vise at $2^n > n$ for $n \geq 0$.

Løsning (a)

Induksjonsstarten er å vise at $\frac{6^n - 1}{5}$ er et heltall for $n = 0$:

$$\frac{6^n - 1}{5} = \frac{6^0 - 1}{5} = \frac{1 - 1}{5} = 0 \text{ er et heltall}$$

Løsning (a)

Induksjonsskrittet er å vise at hvis $\frac{6^k - 1}{5}$ er et heltall, så er $\frac{6^{k+1} - 1}{5}$ et heltall.

- Vi antar at $\frac{6^k - 1}{5}$ er et heltall.
- Vi skal komme frem til at $\frac{6^{k+1} - 1}{5}$ er et heltall.
- La oss gange hele uttrykket med 6.
- Da får vi også et heltall, $\frac{6(6^k - 1)}{5}$.
- Da må $\frac{6^{k+1} - 6}{5}$ være et heltall.
- Men, $\frac{6^{k+1} - 6}{5} = \frac{6^{k+1} - 1}{5} - \frac{5}{5}$.
- Da må også $\frac{6^{k+1} - 1}{5}$ være et heltall

Løsning (b)

“Forklar problemet ved å vise at $2^n > n$ for $n \geq 0$ ved induksjon hvor $n = 0$ er induksjonsstarten.”

- La oss forsøke!
- Induksjonsstarten er å vise at $2^n > n$ holder for $n = 0$.
- Det er greit, for vi har $2^0 = 1 > 0$
- Induksjonsskrittet begynner med å anta at $2^n > n$ holder for $n = k$.
- Induksjonshypotesen er derfor at $2^k > k$.
- Vi må vise at $2^n > n$ holder for $n = k + 1$, det vil si at $2^{k+1} > k + 1$.
- Ved å multiplisere med 2 på begge sider, får vi $2^{k+1} > 2k$.
- Hvis $2k \geq k + 1$ holder, så er vi i mål, siden $2^{k+1} > 2k \geq k + 1$.
- Men, $2k \geq k + 1$ holder kun hvis $k > 0$!
- (For hvis $k = 0$, så vil $2k = 0$ og $k + 1 = 1$.)

Løsning (c)

“Bruk induksjon på en annen måte for å vise at $2^n > n$ for $n \geq 0$.”

- Induksjonsstarten er den samme: $2^0 = 1 > 0$
- Vi viser induksjonsskrittet på en annen måte.
- Vi antar på samme måte at $2^n > n$ holder for $n = k$.
- Vi må vise at $2^n > n$ holder for $n = k + 1$, det vil si at $2^{k+1} > k + 1$.
- Induksjonshypotesen gir at $2^k > k$.
- I stedet for å gange med 2 på begge sider, kan vi *summere* med 2^k på begge sider.
- Vi får da $2^k + 2^k > k + 2^k$.
- Vi har at $2^k + 2^k = 2 \cdot 2^k = 2^{k+1}$, så $2^{k+1} > k + 2^k$.
- Vi har at $2^k \geq 1$, så $k + 2^k \geq k + 1$.
- Vi får dermed at $2^{k+1} > k + 1$.

Oppgave 7.23

Løs følgende rekurrenslikninger

(a) $t(n) - 5t(n - 1) + 4t(n - 2) = 0$, $t(1) = 2$, $t(2) = -1$

Løsning

- Den karakteristiske likningen til likningen er $x^2 - 5x + 4 = 0$.

- Vi kan skrive $(x - 1)(x - 4) = 0$.
- $x = 1$ og $x = 4$ er løsninger av denne likningen.
- Den generelle løsningen til rekurrenslikningen er $t(n) = A \cdot 1^n + B \cdot 4^n$
- $t(1) = 2$ gir $A + 4B = 2$
- $t(2) = -1$ gir $A + 16B = -1$
- Vi får $2 - 4B = -1 - 16B$ og $3 = -12B$ og $B = -\frac{1}{4}$
- Ved å sette inn $B = -\frac{1}{4}$ i $A + 4B = 2$ får vi $A = 3$.
- Løsningen er derfor $t(n) = 3 + (-\frac{1}{4})4^n$

Plenumsregning 9

Diverse ukeoppgaver

Roger Antonsen - 10. april 2008

Oppgaver fra forelesningene

Oppgave (fra forelesningen 10/3).

- a) Ved å bruke den rekursive definisjonen av PL, vis hvordan vi skritt for skritt kan finne verdiene av
- PL(e, d)
 - PL(a, d)
 - PL(ab, d)
 - PL(aba, d)

Husk at *variablene* a og b kan stå for hvilken som helst av *bokstavene* a, b og d.

- b) Vis, ved å bruke definisjonen av R og egenskapen til PL, at $R(abac) = caba$.

Definisjonene av PL og R

Definisjon.

- $PL(e, a) = a$
- $PL(wb, a) = PL(w, a)b$

$PL(w, a)$ blir ordet aw , for alle ord w . Vi viser det induksjon etterpå, som en øvelse.

Definisjon.

- $R(e) = e$
- $R(wb) = PL(R(w), b)$

$R(w)$ blir speilvendingen av ordet w .

Løsning

- a)
- $PL(e, d) = d$
 - $PL(a, d) = PL(ea, d) = PL(e, d)a = da$
 - $PL(ad, d) = PL(a, d)d = dad$
 - $PL(aba, d) = PL(ab, d)a = PL(a, d)ba = PL(e, d)aba = daba$

b)

$$\begin{aligned}R(abac) &= PL(R(aba), c) \\ &= cR(aba) = cPL(R(ab), a) \\ &= caR(ab) = caPL(R(a), b) \\ &= cabR(a) = cabPL(R(e), a) \\ &= cabaR(e) = caba\end{aligned}$$

Oppgave (Øvelse i induksjonsbevis)

Vis ved induksjon på lengden av ord at $PL(w, a) = aw$.

Løsning

- Induksjonsstarten er at påstanden holder når $w = e$.
 - Da er $PL(w, a) = PL(e, a) = a = aw$.
- Induksjonsskrittet:
 - Anta at påstanden holder for ord w av lengde n .
 - Det er vår induksjonshypotese.
 - Anta at vi har et ord wb av lengde $n + 1$, hvor b er en bokstav.
 - Vi må vise at: $PL(wb, a) = awb$
 - Vi har $PL(wb, a) = PL(w, a)b$ ved definisjonen av PL .
 - Ved induksjonshypotesen har vi $PL(w, a) = aw$.
 - Da har vi $PL(wb, a) = PL(w, a)b = awb$.

Oppgave fra forelesningen 12/3

La HF og f være definert som i eksemplet over, og la $S : HF \rightarrow HF$ være definert ved

$$S(X) = X \cup \{X\}.$$

- a) Vis at $X \in HF \Rightarrow S(X) \in HF$.
- b) La $X, Y \in HF$.
Vis at $X \in Y \Rightarrow f(X) < f(Y)$.
- c) Forklar hvorfor det ikke finnes noen $X \in HF$ slik at $X \in X$.
- d) La $N \subseteq HF$ være den minste mengden slik at $\emptyset \in N$ og slik at hvis $X \in N$ så vil $S(X) \in N$.
Drøft sammenhengen mellom N og \mathbb{N}_0 .

Definisjonen av HF (fra forelesningen)

- Vi definerer de hereditært endelige mengdene HF som den minste mengden slik at
 - $\emptyset \in HF$
 - Hvis $X = \{a_1, \dots, a_n\}$ er en endelig delmengde av HF, så er $X \in HF$.
- Vi definerer $f : HF \rightarrow \mathbb{N}_0$ ved rekursjon over HF ved
 - $f(\emptyset) = 0$
 - $f(\{a_1, \dots, a_n\}) = 2^{f(a_1)} + \dots + 2^{f(a_n)}$.

Her må vi anta at mengden er beskrevet uten gjentakelse, det vil si at alle a_i 'ene er forskjellige.

Eksempel.

$\emptyset \in HF, \{\emptyset\} \in HF, \{\emptyset, \{\emptyset\}\} \in HF, \{\{\emptyset\}\} \in HF, \dots$

Husk at $S : HF \rightarrow HF$ er definert ved $S(X) = X \cup \{X\}$.

Løsning (a)

Vis at $X \in HF \Rightarrow S(X) \in HF$.

- Anta at $X \in HF$.
- Da må $X = \{a_1, \dots, a_n\}$ være en endelig delmengde av HF (fordi HF er definert som den *minste* mengden som oppfyller de to kravene).
- Da er $S(X) = X \cup \{X\} = \{a_1, \dots, a_n, X\}$, som også er en endelig delmengde av HF.
- Da må $S(X) \in HF$.

Husk at $f(\emptyset) = 0$ og $f(\{a_1, \dots, a_n\}) = 2^{f(a_1)} + \dots + 2^{f(a_n)}$.

Løsning (b)

La $X, Y \in HF$. Vis at $X \in Y \Rightarrow f(X) < f(Y)$.

- Anta at $Y = \{a_1, \dots, a_n, X\}$.
- Da vil $f(Y) = f(\{a_1, \dots, a_n, X\}) = 2^{f(a_1)} + \dots + 2^{f(a_n)} + 2^{f(X)}$
- Siden $f(X) < 2^{f(X)} \leq f(Y)$, har vi $f(X) < f(Y)$.

Løsning (c)

Forklar hvorfor det ikke finnes noen $X \in HF$ slik at $X \in X$

- Hvis $X \in X$, så gir (b) at $f(X) < f(X)$, men det er umulig.

Løsning (d)

La $N \subseteq HF$ være den minste mengden slik at $\emptyset \in N$ og slik at hvis $X \in N$ så vil $S(X) \in N$. Drøft sammenhengen mellom N og \mathbb{N}_0 .

- N og \mathbb{N}_0 er essensielt like strukturer. Vi kan definere en en-til-en-korrespondanse mellom elementene på følgende måte.

$$\begin{aligned} 0 &\leftrightarrow \emptyset &= \emptyset \\ 1 &\leftrightarrow S(\emptyset) &= \{\emptyset\} \\ 2 &\leftrightarrow S(S(\emptyset)) &= \{\emptyset, \{\emptyset\}\} \\ 3 &\leftrightarrow S(S(S(\emptyset))) &= \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\} \end{aligned}$$

- Mer presist, la $s : \mathbb{N}_0 \rightarrow N$ være definert ved rekursjon over \mathbb{N}_0 på følgende måte.
 - $s(0) = \emptyset$
 - $s(n+1) = S(s(n))$
- Det vil si, $s(n)$ er resultatet av å anvende S n ganger på \emptyset .
- Denne funksjonen er surjektiv og injektiv.

Løsninger på ekstraoppgavene til uke 13

Oppgave - Ekstraoppgave 1 til uke 13

La $f(w)$ være summen av antall forekomster av a og b i w minus to ganger antall forekomster av c.

- a) Vis hvordan vi kan definere f ved rekursjon på oppbyggingen av w .
- b) Finn en pseudokode for en algoritme som beregner f .

Løsning (Ekstraoppgave 1a til uke 13)

- $f(e) = 0$
- $f(aw) = 1 + f(w)$
- $f(bw) = 1 + f(w)$
- $f(cw) = 2 + f(w)$

Løsning (Ekstraoppgave 1b til uke 13)

1. Input n
2. Input $w = v_1, \dots, v_n$
3. $sum \leftarrow 0$
4. **For** $i = 1$ **to** n **do**
 - 4.1. **If** $v_i = a$ **or** $v_i = b$ **then**
 - 4.1.1. $sum \leftarrow sum + 1$
 - else** [her vil $v_i = c$]
 - 4.1.2. $sum \leftarrow sum + 2$
5. Output sum

Løsning (Ekstraoppgave 1b til uke 13)

Det er fristende å skrive denne litt annerledes.

1. Input w
2. $sum \leftarrow 0$
3. **While** $w \neq e$ **do** [e er det *tomme* ordet]
 - 3.1. $x \leftarrow$ første bokstav i w [x er enten a, b eller c]
 - 3.2. $r \leftarrow$ resten av w

3.3. **If $x = a$ or $x = b$ then**

3.3.1. $sum \leftarrow sum + 1$

else [her vil $x = c$]

3.3.2. $sum \leftarrow sum + 2$

3.4. $w \leftarrow r$

4. Output sum

Oppgave - Ekstraoppgave 2 til uke 13

La A være et utsagn på svak normalform, hvor vi antar at vi bare bruker utsagnsvariablene p , q og r .

Vi lar A^* være formelen vi får ved å erstatte alle forekomster av p , q og r med $\neg p$, $\neg q$ og $\neg r$, alle forekomster av \wedge med \vee og alle forekomster av \vee med \wedge .

- Vis hvordan vi kan definere A^* ved rekursjon på oppbyggingen av A (husk at vi har et tilfelle $A = \neg B$ også).
- Vis ved induksjon at

$$A^* \Leftrightarrow \neg A$$

Løsning (Ekstraoppgave 2a til uke 13)

En rekursiv definisjon av $*$:

- $v^* = \neg v$, når v er en utsagnsvariabel
- $(B \wedge C)^* = (B^* \vee C^*)$
- $(B \vee C)^* = (B^* \wedge C^*)$
- $(\neg B)^* = \neg(B^*)$

Løsning (Ekstraoppgave 2b til uke 13)

Viser at $A^* \Leftrightarrow \neg A$ ved induksjon på formler på svak normalform.

- Induksjonsstart er at $A^* \Leftrightarrow \neg A$ holder når A er en utsagnsvariabel.
 - Vi viser først \Rightarrow -retningen:
 - Anta at v^* er sann, hvor v er en utsagnsvariabel
 - Siden $v^* = \neg v$, må $\neg v$ også være sann.
 - Vi viser så \Leftarrow -retningen:

- Anta at $\neg v$ er sann, hvor v er en utsagnsvariabel
- Siden $\neg v = v^*$, må v^* også være sann.

Løsning

- Induksjonsskrittet er tredelt, for hvis A ikke er en utsagnsvariabel, så er A enten på formen $B \wedge C$, $B \vee C$ eller $\neg B$.
- I alle tilfellene er *induksjonshypotesen* at påstanden holder for B og C .
- Induksjonshypotesen sier at $B^* \Leftrightarrow \neg B$ og $C^* \Leftrightarrow \neg C$ holder.
- Vi viser først at påstanden holder hvis A er på formen $(B \wedge C)$.

Løsning

- Vi viser først \Rightarrow -retningen:
 - Anta at $(B \wedge C)^*$ er sann.
 - Da er $B^* \vee C^*$ sann, ved definisjonen av $*$.
 - Da er enten B^* eller C^* sann, ved sannhetsbetingelsen for \vee -formler.
 - Da er enten $\neg B$ eller $\neg C$ sann, ved induksjonshypotesen.
 - Da er ikke både B og C sanne, ved sannhetsbetingelsen for \neg -formler.
 - Da er $\neg(B \wedge C)$ sann, ved sannhetsbetingelsen for \wedge -formler.
- Vi viser så \Leftarrow -retningen:
 - Anta at $\neg(B \wedge C)$ sann.
 - Da er ikke både B og C sanne, ved sannhetsbetingelsen for \wedge -formler.
 - Da er enten $\neg B$ eller $\neg C$ sann, ved sannhetsbetingelsen for \neg -formler.
 - Da er enten B^* eller C^* sann, ved induksjonshypotesen.
 - Da er $B^* \vee C^*$ sann, ved sannhetsbetingelsen for \vee -formler.
 - Da er $(B \wedge C)^*$ sann, ved definisjonen av $*$.

Løsning

- Vi viser så at påstanden holder hvis A er på formen $(B \vee C)$.
 - Denne gangen viser vi begge retninger samtidig via en rekke ekvivalenser.

$$\begin{aligned}
 (B \vee C)^* &\Leftrightarrow B^* \wedge C^* \text{ (ved definisjon av } *) \\
 &\Leftrightarrow \neg B \wedge \neg C \text{ (ved IH og sannhetsbetingelsen for } \wedge\text{-formler)} \\
 &\Leftrightarrow \neg(B \vee C) \text{ (ved sannhetsbetingelsene for } \vee\text{- og } \neg\text{-formler)}
 \end{aligned}$$

- Hvis A er på formen $\neg B$ får vi:

$$\begin{aligned}(\neg B)^* &\Leftrightarrow \neg(B^*) \text{ (ved definisjon av } *) \\ &\Leftrightarrow \neg(\neg B) \text{ (ved induksjonshypotesen)}\end{aligned}$$

Oppgave - Ekstraoppgave 3 til uke 13

La U være mengden av utsagnslogiske formler i utsagnsvariablene p , q og r , hvor vi bare bruker bindeordene \neg , \wedge og \vee .

Dette er det formelle språket som er beskrevet på forelesningene.

- Finne en rekursiv definisjon av funksjonen f hvor $f(A)$ er antall parenteser i A .
- Skriv ut argumentet for at $f(A)$ alltid er et partall som et induksjonsbevis.

Løsning (Ekstraoppgave 3a til uke 13)

- $f(v) = 0$, hvor v er en utsagnsvariabel
- $f(\neg B) = f(B)$
- $f((B \wedge C)) = 2 + f(B) + f(C)$
- $f((B \vee C)) = 2 + f(B) + f(C)$

Løsning (Ekstraoppgave 3b til uke 13)

- Vi viser ved induksjon over formler at $f(A)$ er et partall.
- Induksjonsstart er at $f(v)$, hvor v er en utsagnsvariabel, er et partall.
- Det holder fordi $f(v) = 0$.
- Induksjonsskrittet er igjen tredelt.
- Vi viser tilfellet for $A = (B \wedge C)$ og tar eventuelt resten på tavla.
- Hvis $A = (B \wedge C)$, får vi $f(A) = f((B \wedge C)) = 2 + f(B) + f(C)$.
- Ved induksjonshypotesen er $f(B)$ og $f(C)$ partall.
- Da må $f(A)$ også være et partall.

Oppgave - Ekstraoppgave 4 til uke 13

Vi kan definere mengden av korrekte uttrykk med to sett $(,)$ og $[,]$ av parenteser ved følgende kortform

- $P ::= e|(P)||[P]||PP$

- Formuler denne definisjonen som en induktiv definisjon uten bruk av denne spesielle notasjonen, dvs. bruk formuleringen "den minste mengden slik at".
- Formuler, etter beste evne, et kriterium for når et ord med symbolene $(,)$, $[$ og $]$ er korrekt, slik at kriteriet kan brukes som grunnlag for en algoritme som tester dette.
- Finn en pseudokode for en algoritme som tester om et ord er et korrekt parentesuttrykk i denne betydningen.

Hint: Du trenger en hjelpevariabel som skal "huske" hva som mangler for at en del av ordet skal bli korrekt. Du vil sannsynligvis måtte la denne variabelen ta ord i et egnet alfabet som verdier.

Løsning (Ekstraoppgave 4a til uke 13)

La X være den minste mengden slik at følgende holder:

- $e \in X$
- $P \in X \Rightarrow (P) \in X$
- $P \in X \Rightarrow [P] \in X$
- $P \in X$ og $Q \in X \Rightarrow PQ \in X$

Løsning (Ekstraoppgave 4b til uke 13)

- X består av ord slik at parentesene er *balanserte*.
- Et kriterium for når et ord P er korrekt er følgende:
 - Anta at vi leser P fra venstre mot høyre.
 - Samtidig som vi leser ordet lager vi et nytt ord Q som består av kun venstreparenteser.
 - Hver gang vi treffer en venstreparentes legger vi denne til Q .
 - Da blir Q til enten $Q($ eller $Q[$.
 - Hver gang vi treffer en høyreparentes sjekker vi følgende.
 - Hvis Q er tom, så er ikke P korrekt. Avslutt.
 - Hvis høyreparentesen er av samme type som den siste venstreparentesen i Q , så fjern denne venstreparentesen fra Q og fortsett.

- Hvis den ikke er samme type, så er ikke P korrekt. Avslutt.
- Hvis vi klarer å lese alle parentesene på denne måten, og Q er tom til slutt, så er P korrekt.
- Hvis Q ikke er tom, så er ikke P godkjent.

Repetisjon av rekurrenslikninger

- Vi repeterer raskt alle definisjonene og teoremene fra forelesningene om rekurrenslikninger.
- Vi diskuterer eventuelle vanskeligheter og regner på tavla hvis det trengs.
- Deretter løser vi noen av oppgavene fra boka.

Oppgave 7.23

Løs følgende rekurrenslikning

$$(b) \quad t(n) - 2t(n-1) - 8t(n-2) = 0, \quad t(1) = 1, \quad t(2) = 0$$

- Vi løser denne, og tar eventuelt flere på tavla.

Løsning (b)

- Den karakteristiske likningen til likningen er $x^2 - 2x - 8 = 0$.
- Vi kan skrive $(x + 2)(x - 4) = 0$.
- $x = -2$ og $x = 4$ er løsninger av denne likningen.
- Den generelle løsningen til rekurrenslikningen er $t(n) = A \cdot (-2)^n + B \cdot 4^n$
- $t(1) = 1$ gir $-2A + 4B = 1$
- $t(2) = 0$ gir $4A + 16B = 0$
- Fra $-2A + 4B = 1$ får vi $4A = 8B - 2$, og dermed $8B - 2 + 16B = 0$, som gir $24B = 2$ og $B = \frac{1}{12}$
- Ved å sette inn $B = \frac{1}{12}$ i $-2A + 4B = 1$ får vi $-2A + \frac{1}{3} = 1$ og $A = -\frac{1}{3}$.
- Løsningen er derfor $t(n) = -\frac{1}{3} \cdot (-2)^n + \frac{1}{12} \cdot 4^n$.

Plenumsregning 10

Diverse ukeoppgaver

Roger Antonsen - 17. april 2008

Vi øver oss litt på løse rekurrenslikninger.

Oppgave 7.23

Løs følgende rekurrenslikning

$$(c) \quad t(n) - 6t(n-1) + 9t(n-2) = 0, \quad t(1) = 3, \quad t(2) = 2$$

Løsning (c)

- Den karakteristiske likningen til likningen $t(n) - 6t(n-1) + 9t(n-2) = 0$ er $x^2 - 6x + 9 = 0$.
- Vi kan skrive $(x-3)(x-3) = 0$.
- $x = 3$ er den eneste løsningen av denne likningen.
- Den generelle løsningen til rekurrenslikningen er $t(n) = (A + Bn) \cdot 3^n$.
- $t(1) = 3$ gir $(A + B \cdot 1)3^1 = 3A + 3B = 3$.
- $t(2) = 2$ gir $(A + B \cdot 2)3^2 = 9A + 18B = 2$.
- Fra $3A + 3B = 3$ får vi $9A + 9B = 9$, og dermed $18B - 2 = 9B - 9$, som gir $9B = -7$ og $B = -\frac{7}{9}$.
- Ved å sette inn $B = -\frac{7}{9}$ i $9A + 9B = 9$ får vi $9A - 7 = 9$ og $9A = 16$, som gir $A = \frac{16}{9}$.
- Løsningen er derfor $t(n) = (\frac{16}{9} - \frac{7}{9}n)3^n$.

Oppgave 9.1

I en klasse på 35 elever, så snakker 12 tysk og 5 japansk. Hvis to elever snakker begge disse språkene, hvor mange studenter snakker ingen av språkene.

Løsning

- La T være mengden av de som snakker tysk.
- La J være mengden av de som snakker japansk.

- Vi har $|T| = 12$ og $|J| = 5$ og $|T \cap J| = 2$.
- Inklusjons- og eksklusjonsprinsippet sier: $|T \cup J| = |T| + |J| - |T \cap J|$.
- Vi får derfor $|T \cup J| = 12 + 5 - 2 = 15$.
- Det betyr at 15 av 35 snakker enten tysk eller japansk.
- Da er det 20 som ikke snakker noen av språkene.

Oppgave 9.8

Lovlige passord på 8 tegn. Første tegn må være bokstav, og resten må være bokstaver eller tall. (Vi skiller ikke mellom store og små bokstaver og tar ikke med æ,ø,å.) Et passord må inneholde minst ett tall. Hvor mange mulige passord fins det?

Løsning

- Antall muligheter for første tegn er 26.
- Vi regner ut mulige kombinasjoner for de 7 neste tegnene først. Også ganger vi med 26 til slutt.
- Minst ett av de 7 tegnene må være et tall.
- Antall strenger på 7 tegn som ikke inneholder tall er 26^7 .
- Antall strenger på 7 tegn er til sammen 36^7 .
- Antall muligheter for de neste 7 tegnene er derfor $36^7 - 26^7$.
- Svaret er derfor $26 \cdot (36^7 - 26^7)$, som er 1828641201920.

Oppgave 9.11

Regn ut (a) ${}^{10}P_6$, (b) ${}^{12}P_8$, (c) ${}^{13}C_5$, og (d) $\binom{15}{11}$.

Løsning

$${}^{10}P_6 = \frac{10!}{(10-6)!} = 10 \cdot 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 = 151200$$

$${}^{12}P_8 = 12 \cdot 11 \cdot 10 \cdot 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 = 19958400$$

$${}^{13}C_5 = \frac{13!}{8!5!} = \frac{13 \cdot 12 \cdot 11 \cdot 10 \cdot 9}{5 \cdot 4 \cdot 3 \cdot 2} = 13 \cdot 11 \cdot 9 = 1287$$

$$\binom{15}{11} = \frac{15 \cdot 14 \cdot 13 \cdot 12}{4 \cdot 3 \cdot 2} = 15 \cdot 7 \cdot 13 = 1365$$

Oppgave 9.13

Poengsummene til 12 elever skal gis som data til et program.

15, 13, 18, 15, 7, 12, 10, 13, 9, 5, 15, 17

I hvor mange forskjellige rekkefølger kan dette gjøres?

Løsning

- Vi sorterer dem for å se hvor mange like tall det er:

5, 7, 9, 10, 12, 13, 13, 15, 15, 15, 17, 18

- 13 forekommer 2 ganger, og 15 forekommer 3 ganger.
- Svaret blir

$$\frac{12!}{2!3!} = 11! = 39916800$$

Oppgave 9.17

Bevis følgende påstand

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

Løsning

Tavleregning.

Oppgave 9.18

Vis at ${}^n P_r = n({}^{n-1} P_{r-1})$.

Løsning

$$n \cdot ({}^{n-1} P_{r-1}) = n \cdot \frac{(n-1)!}{(n-1-(r-1))!} = \frac{n(n-1)!}{(n-r)!} = \frac{n!}{(n-r)!} = {}^n P_r$$

Oppgave 9.20

Ved å tolke $\binom{n}{r}$ som antall delmengder av størrelse r av en mengde av størrelse n , finn verdien til

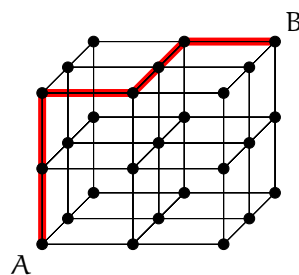
$$\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \dots + \binom{n}{n}$$

Løsning

Summen må bli antall delmengder totalt, som er 2^n .

Oppgave fra forelesning

På hvor mange forskjellige måter kan vi gå fra A til B i denne $2 \times 2 \times 2$ -kuben? Vi må gå ett steg av gangen og kun oppover, til høyre eller innover?



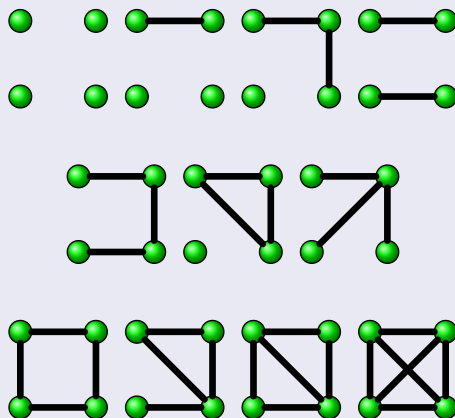
Løsning

- Vi kan representere mulige måter å gå fra A til B som antall forskjellige ord i alfabetet $\{\uparrow, \rightarrow, \nearrow\}$
- Det er $6!$ måter å stokke om tegnene i et ord som er 6 tegn langt.
- Av disse må vi identifisere ord som er like.
- Svaret blir derfor $\frac{6!}{2!2!2!} = 6 \cdot 5 \cdot 3 = 90$.
- Det går an å finne svaret på en annen måte også.
- Ordet skal være seks tegn langt, og på nøyaktig to av posisjonene må tegnet \uparrow stå.
- Det er $\binom{6}{2} = 15$ forskjellige måter å gjøre det på.
- Da er det fire posisjoner igjen, og på nøyaktig to av disse må \rightarrow stå.
- Det er $\binom{4}{2} = 6$ måter å gjøre det på.
- Det er altså $15 \cdot 6 = 90$ forskjellige ord av lengde 6.

Oppgave 10.2

Tegn alle 11 enkle grafer med 4 noder.

Løsning



Oppgave 10.3

Bekreft for hver av grafene i figuren at summen av gradene til nodene er lik det dobbelte av antall kanter.

Løsning

Tavleregning.

Oppgave 10.4

Tegn en graf hvor nodene har følgende grader, eller forklar hvorfor det ikke fins noen slik graf.

- (a) 2, 3, 3, 4, 5
- (b) 2, 3, 3, 3, 3

Løsning

- (a) Summen av gradene er $2 + 3 + 3 + 4 + 5 = 17$. Det er umulig, siden summene av gradene må være et partall.
- (b) Summene av gradene er $2 + 3 + 3 + 3 + 3 = 14$. Grafen må ha 5 noder og 7 kanter. Vi tegner på tavla.

Oppgave 10.5

Finn et alternativt bevis for at en komplett graf med n noder har $[n(n - 1)/2]$ kanter, ved å bruke resultatet om at summen av gradene til nodene er lik to ganger antallet kanter.

Løsning

- La K_n være den komplette grafen med n noder.
- Hva er graden til en node i denne grafen? Graden må være $n - 1$ for alle n nodene.
- Summen til gradene blir derfor $n(n - 1)$.
- Resultatet sier at dette er to ganger antallet kanter.
- Antallet kanter blir derfor $[n(n - 1)/2]$. (Som er det samme som $\binom{n}{2}$.)

Oppgave 10.6

Hvis en enkel graf G har n noder og m kanter, hvor mange kanter har \overline{G} ?

Løsning

- Vi vet at den komplette grafen med n noder har $\binom{n}{2}$ kanter.
- Vi har at e er en kant i \overline{G} hvis og bare hvis e ikke er en kant i G .
- Hvis vi tar kantene fra G og kantene fra \overline{G} , så får vi kantene i den komplette grafen, som det er $\binom{n}{2}$ av.
- Antall kanter i \overline{G} blir derfor $\binom{n}{2} - m$.

Oppgave 10.7

La koblingsmatriser for grafene i figuren

Løsning

Tavleregning.

Oppgave 10.8

Tegn grafene med koblingsmatrisene som gitt i oppgaven.

Løsning

Tavleregning.

Plenumsregning 11

Ukeoppgaver fra kapittel 10 & Induksjonsbevis

Roger Antonsen - 24. april 2008

Grafteori

Vi regner oppgavene på tavlen i dag.

- Oppgave 10.9
- Oppgave 10.10
- Oppgave 10.11
- Oppgave 10.12
- Oppgave 10.13
- Oppgave 10.14
- Oppgave 10.15
- Oppgave 10.17
- Oppgave 10.18
- Oppgave 10.19
- En nøtt om en maur og en kube.

Induksjonsbevis

- Bevis ved induksjon er et kraftig verktøy som vi kan bruke til å bevise *universelle* påstander.
- En *universell påstand* er en påstand på formen "For enhver $x \in S$ så er det slik at...".
- Typiske muligheter for mengden S er
 - \mathbb{N}
 - en mengde av ord i et alfabet (en *induktiv definert mengde*),
 - en mengde av utsagnslogiske formler,
 - en mengde av grafer,
 - en mengde av trær,
 - og mye, mye mer...
- Noen typiske måter å bevise en universell påstand på er følgende.
- Man kan velge et vilkårlig element fra S og vise at påstanden holder for dette elementet. Siden elementet er *vilkårlig* valgt, så ligger det ingen føringer på hvilket element i S som er valgt, og hvis påstanden holder for dette elementet, så må påstanden holde for *alle* elementene i S .

- Man kan gi et “ikke-konstruktivt” bevis ved å anta (for motsigelse) at påstanden *ikke* holder for alle $x \in S$ og så utlede en motsigelse.
- Man kan gi et induksjonsbevis.

Følgende er sakset fra forelesning 14:

Definisjon.

La $P(n)$ være et predikat med en variabel n for et element i \mathbb{N} .

Anta at vi kan bevise

1. $P(1)$
2. $\forall n(P(n) \rightarrow P(n + 1))$

Da kan vi konkludere $\forall nP(n)$.

Denne måten å bevise $\forall nP(n)$ på kalles induksjon.

1. kalles for *induksjonsstarten* eller *basissteget*
2. kalles for *induksjonsskrittet* eller *induksjonssteget*

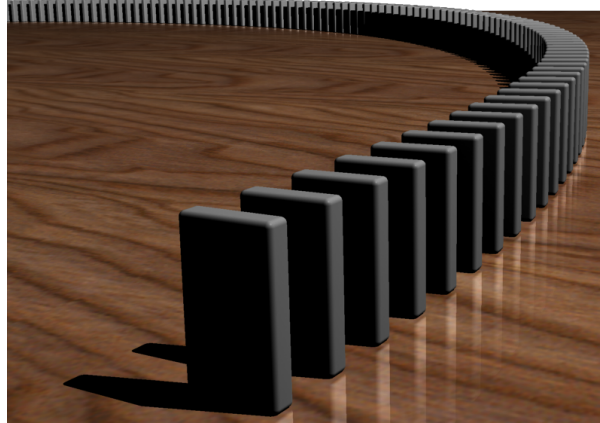
Viktig å huske på!

- Man må alltid ha klart for seg hva som bevises i et induksjonsbevis.
- Man må vite hva *PÅSTANDEN* som bevises er.
- Først så vises *PÅSTANDEN* for tallet 1.
- Så antar man at *PÅSTANDEN* holder for n .
- Dette at *PÅSTANDEN* holder for n kalles gjerne for *induksjonshypotesen*.
- Ut fra *denne antakelsen* - altså antakelsen om at induksjonshypotesen holder - så viser man at *PÅSTANDEN* holder for $n + 1$.

Jeg gjentar (i tilfelle noen ikke fikk det med seg):

- Man må alltid ha klart for seg hva *PÅSTANDEN* som bevises er.

Tenk domino!



- Induksjonsstarten er at første brikke faller.
- Induksjonsskrittet er at en brikke slår den neste over ende.
- “Ved induksjon vil alle brikkene falle over ende.”

Flere viktige ting å huske på

- Man må alltid vise både induksjonsstarten og induksjonsskrittet.
 - Begynn induksjonsskrittet med å anta at induksjonshypotesen holder, f.eks. at påstanden holder for et tall n .
 - Bruk ordet “anta”. Hele poenget med matematiske bevis er å tenke ut fra antakelser!
 - Hvis du ikke har brukt induksjonshypotesen underveis, så er det sannsynligvis noe galt et sted.
 - Avslutt et induksjonsbevis med å si noe som likner på “Ved induksjon følger det at...”.
 - Les læreboka og forelesningsnotatene.
 - Les gjerne på nettet: http://en.wikipedia.org/wiki/Mathematical_induction
 - Og øv!
-
- La oss bevise at påstanden

$$1 + 3 + 5 + \dots + (2n - 1) = n^2$$

holder for alle naturlige tall $n \geq 1$.

- Først sjekker vi om det er plausibelt:
 - $n = 1$ gir $1 = 1^2$
 - $n = 2$ gir $1 + 3 = 4 = 2^2$
 - $n = 3$ gir $1 + 3 + 5 = 9 = 3^2$
 - $n = 4$ gir $1 + 3 + 5 + 7 = 16 = 4^2$
 - og så videre...
- Men, vi ønsker et bevis for at dette holder for *alle* naturlige tall!
- Da er et induksjonsbevis veien å gå.
- Vi viser påstanden ved induksjon på naturlige tall.

Bevis

- Hva er *PÅSTANDEN*? Hva vil det si at påstanden holder for n ?
- Det er at $1 + 3 + 5 + \dots + (2n - 1) = n^2$ er sant.
- Induksjonsstarten er at *PÅSTANDEN* holder for $n = 1$.
 - Vi setter inn 1 for n og får $1 = 1^2$, som stemmer.
- Induksjonsskrittet er at *hvis* *PÅSTANDEN* holder for k , så holder den for $k + 1$.
- Her kommer induksjonsskrittet i beviset.
 - Anta at *PÅSTANDEN* holder for k .
 - Det betyr at $1 + 3 + 5 + \dots + (2k - 1) = k^2$ er sant.
 - Nå må vi vise at *PÅSTANDEN* holder for $k + 1$.
 - Vi må altså vise at $1 + 3 + 5 + \dots + (2(k + 1) - 1) = (k + 1)^2$ er sant.
 - Vi må vise at venstresiden er lik høyresiden.
 - Venstresiden er lik $1 + 3 + 5 + \dots + (2k + 1)$ som er lik $1 + 3 + 5 + \dots + (2k - 1) + (2k + 1)$
 - Vi må vise at dette er lik høyresiden, som er lik $(k + 1)^2$.
- Vi må tenke ut fra *antakelser*!
- Hva vet vi?
 - Vi vet at $1 + 3 + 5 + \dots + (2k - 1) = k^2$ er sant.
(Det er at *PÅSTANDEN* holder for k .)
- Hva er målet vårt?
 - Det er å vise at *venstresiden* $1 + 3 + 5 + \dots + (2k - 1) + (2k + 1)$ er lik *høyresiden* $(k + 1)^2$.
(Det er at *PÅSTANDEN* holder for $k + 1$.)
- Ved å bytte ut $1 + 3 + 5 + \dots + (2k - 1)$ med k^2 i *venstresiden*, så får vi $k^2 + (2k + 1)$.
- Det er lik $(k + 1)^2$
som er det samme som høyresiden, *nøyaktig* det vi skulle vise!
- Ved induksjon følger det at påstanden holder for alle naturlige tall $n \geq 1$.
- Induksjonsbevis for universelle påstander om andre ting enn naturlige tall er også veldig viktig.
- Det er viktig å tenke over hva som er induksjonsstarten og hva som er induksjonsskrittet i de forskjellige tilfellene.
- Hva som er induksjonsstarten og induksjonsskrittet samsvarer gjerne med hvordan mengden er definert.
- F.eks. har vi induksjonsbevis for universelle påstander om *ord som er induktivt definert*.
 - En mengde av ord er ofte definert som den *minste* mengden av strenger slik at *den tomme strengen er med* og slik at *hvis X er med, så er også Y med*. (Hvor X og Y er passende beskrivelser.)
 - Da blir induksjonsstarten at påstanden holder for den tomme strengen.
 - Og induksjonsskrittet vil svare til hvordan mengden er definert.

Superenkelt eksempel

- La S være den minste mengden slik at
 - sifrene 0 og 1 er med i S , og
 - hvis a er med i S og det siste sifferet i a er 0, så er $a1$ med i S , og
 - hvis a er med i S og det siste sifferet i a er 1, så er $a0$ med i S .
- Oppgave: vis at det ikke fins noe ord i S hvor det samme sifferet forekommer rett etter hverandre.
- Legg merke til at dette er en *universell* påstand: For alle ord i S så er det slik at to etterfølgende sifre ikke er like.
- Vi viser *påstanden* ved induksjon på lengden av ord, siden hvert steg i konstruksjonen over øker lengden med 1.
- Dette er en vanlig måte å formulere et induksjonsbevis på.
- Det må da være helt klart hva *lengde* betyr. I dette tilfellet er det antall sifre i ordet.
- Det vanlige er at induksjonsbeviset følger stegene i hvordan en mengde er definert.

- Induksjonsstarten.
 - Vi viser at PÅSTANDEN holder for ord av lengde 1.
 - Et ord av lengde 1 er enten 0 eller 1 og her det ingen etterfølgende sifre som er like.
- Induksjonsskrittet.
 - Anta at PÅSTANDEN holder for ord av lengde n , hvor $n \geq 1$.
 - Vi skal vise at PÅSTANDEN holder for ord av lengde $n + 1$.
 - La a være et ord av lengde $n + 1$.
 - Siden $n \geq 1$ må ordet har minst lengde 2.
 - Hvis siste siffer i a er 0, så må a være lik $b0$, hvor siste siffer i b er 1.
 - Hvis siste siffer i a er 1, så må a være lik $b1$, hvor siste siffer i b er 0.
 - Ved antakelsen (induksjonshypotesen), så inneholder ikke b to etterfølgende sifre som er like.
 - Da gjør ikke a det heller.
- Påstanden følger ved induksjon på lengden av ord.

Vi tar mer repetisjon på tavla

- Vi snakker om andre former for induksjon og regner eventuelt flere oppgaver.
- Vi ser litt på
 - Induksjonsbevis i grafteori.
 - * F.eks. beviset for at ethvert tre med n noder har $n - 1$ kanter.
 - Induksjonsbevis for å vise at rekursivt definerte funksjoner har bestemte egenskaper, f.eks. at de oppfyller bestemte likninger.
 - * Vi viste f.eks. at funksjonen definert ved $f(1) = 1$ og $f(n + 1) = f(n) + (2n - 1)$ oppfyller likningen $f(n) = n^2$.

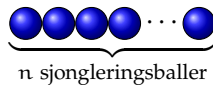
- Andre induksjonsprinsipper, f.eks. når vi antar at en påstand holder for alle tall $m < n$.
- Grunnlaget for induksjonsbevis.

Oppgave: hvor er feilen?

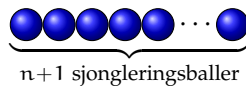
- PÅSTAND: Alle sjongleringsballer har samme farge.
- BEVIS: Ved induksjon på mengden av sjongleringsballer.
- INDUKSJONSSTART: I en mengde sjongleringsballer av størrelse 1, så holder påstanden. Her, f.eks., er alle blå:



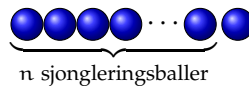
- INDUKSJONSSKRITTET:
- Anta at påstanden holder for alle mengder av størrelse n .



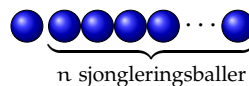
- Dette er induksjonshypotesen vår.
- La S være en mengde sjongleringsballer av størrelse $n + 1$.



- Vi ser på de n første sjongleringsballene i S :



- Denne delmengden har størrelse n . Ved induksjonshypotesen må alle disse n ballene ha samme farge.
- Vi ser på de n siste sjongleringsballene i S :



- Siden dette er en mengde av størrelse n , så må alle ballene ha samme farge.
- Da må alle ballene i S ha samme farge.
- Ved induksjon må alle sjongleringsballer ha samme farge. ELLER?

Plenumsregning 12

Diverse oppgaver

Roger Antonsen - 22. mai 2008

Plan

- Dette er siste plenumsregning.
- Vi regner stort sett eksamensoppgaver.
- Neste uke blir det repetisjon på mandag og onsdag.
- Send epost til Dag eller meg med eventuelle ønsker om hva dere vil ha gjennomgått.
- Bruk orakeltjenestene!
 - Sted: Rom B534 i Niels Henrik Abels hus
 - Tider:
 - * Tirsdag 27/5 12-14
 - * Onsdag 28/5 10-12
 - * Tirsdag 3/6 12-14
 - * Onsdag 4/6 12-14
- Jeg lager et løsningsforslag til ekstraoppgavene (Oppgavesett 4) og legger det ut i begynnelsen av neste uke.

Noen tips til eksamen

- Vær effektive og ikke bli sittende og tenke altfor lenge på hver oppgave.
 - Det vil si, ikke få panikk hvis noe er litt for vanskelig ved første øyekast. Gå heller tilbake til oppgaven litt senere.
- Øv på å løse oppgaver og skrive bevis.
 - Ikke la eksamen være første gangen.
- Les oppgaveteksten nøye og svar på det som det spørres etter.
 - Ja, det er lurt å lese gjennom hele oppgavesettet først.
- Hvis du står fast, vis i hvert fall hva du har forstått. Den som retter en eksamen er ute etter å finne ut hva kandidaten *har forstått*.

Eksamen 12/6-06 Oppgave 2

La X være mengden som består av alle 8-bit strenger med 0-er og 1-ere. For $k \in \mathbb{Z}$, $0 \leq k \leq 8$, definer $X_k = \{s \in X \mid \text{antall 0-er i } s \text{ er lik } k\}$.

(a) Begrunn at

$$|X_k| = \binom{8}{k}$$

- (b) Vi definerer en relasjon R på X ved at sRt hvis og bare hvis antall 0-er i s er kongruent modulo 3 med antall 0-er i t . Du kan ta det for gitt at dette er en ekvivalensrelasjon. La E betegne ekvivalensklassen til $s = 11011011$. Beregn $|E|$.
- (c) Skriv pseudokoden for en algoritme som avgjør om to elementer i X er i relasjon med hverandre eller ikke (med hensyn på relasjonen definert i b)). Input antas å være to 8-bit strenger $s_1 \cdots s_8$ og $t_1 \cdots t_8$, mens output skal være "de to strengene er i relasjon med hverandre" eller "de to strengene er ikke i relasjon med hverandre".

Vi danner oss først et bilde av hva oppgaven snakker om.

- Alle strengene i X består av 8 bit.
- Alt i alt er det derfor $2^8 = 256$ strenger i X , eller skrevet på en annen måte, $|X| = 256$.
- X_0 betegner mengden av strenger uten 0-ere.
- X_1 betegner mengden av strenger med nøyaktig 1 stk. 0-er:

- 01111111
- 10111111
- 11011111
- 11101111
- 11110111
- 11111011
- 11111101
- 11111110

- Vi ser at $|X_1| = 8$.
- X_2 betegner mengden av strenger med nøyaktig 2 stk. 0-ere.
- Oppgave a) spør om $|X_k|$, det vil si antall elementer i X_k .

- Hva spør oppgave a) om? Den spør om en begrunnelse.
- Vi skal begrunne at

$$|X_k| = \binom{8}{k}$$

- Løsningen på a) blir derfor slik:
- En streng i X_k har nøyaktig k antall 0-ere. Siden det er 8 bit i en streng, er antall strenger med k 0-ere lik antallet måter man kan velge k elementer fra en mengde med 8 elementer på. Det er $\binom{8}{k}$ måter å velge k elementer fra en mengde med 8 elementer på. Det er derfor $\binom{8}{k}$ antall 8-bit strenger som har k 0-ere. Derfor er $|X_k| = \binom{8}{k}$. (Vi ser f.eks. at dette stemmer for $|X_0|$, som er lik $\binom{8}{0} = 1$, og $|X_1|$, som er lik $\binom{8}{1} = 8$.)
- Relasjonen R på X er definert ved at sRt hvis og bare hvis antall 0-er i s er kongruent modulo 3 med antall 0-ere i t .
- Hva betyr det?
 - "kongruent modulo 3" betyr at differansen er delelig med 3.

- F.eks. er 1 og 7 kongruente modulo 3, siden $7 - 1 = 6$ er delelig med 3.
- Hva ber oppgave b) om? Den ber oss om å beregne $|E|$, hvor E er ekvivalensklassen til $s = 11011011$.
 - For å finne $|E|$, antallet elementer i E , er det lurt å først finne mengden E .
 - E er mengden av strenger t slik at sRt . (Det er *definisjonen* av en ekvivalensklasse.)
 - For å finne ut hva E er, så må vi vite hvor mange 0-ere det er i s . Det er 2 stk. 0-ere i s .
 - Hvilke t er R -relatert til s ? Vi har sRt når t har et antall 0-ere som er kongruente modulo 3 med 2.
 - Da har vi at $t \in E$ når t har 2, 5 eller 8 stk. 0-ere.
 - Sagt på en annen måte: t er i E hvis t er i X_2, X_5 eller X_8 .
 - På dette tidspunktet skal det ringe en bjelle: vi skal bruke informasjonen fra oppgave a).
- Vi har at $E = X_2 \cup X_5 \cup X_8$.
- Dermed får vi $|E| = |X_2 \cup X_5 \cup X_8| = |X_2| + |X_5| + |X_8|$.
- Vi bruker inklusjons- og eksklusjonsprinsippet her, men siden mengdene ikke er overlappende, er det ikke nødvendig å bruke eksklusjon.
- Vi får at $|E| = \binom{8}{2} + \binom{8}{5} + \binom{8}{8} = 28 + 56 + 1 = 85$.
- Da har vi svart på oppgave b).
- Jeg har riktignok skrevet en god del mer her enn det som trengs for en god besvarelse.
- Oppgave c) ber om en pseudokode for å avgjøre om to elementer i X er i relasjonen R med hverandre eller ikke.

1. Input s, t [$s = s_1 \cdots s_8$ og $t = t_1 \cdots t_8$]
2. $a \leftarrow$ antall 0-ere i s
3. $b \leftarrow$ antall 0-ere i t
4. **If** $(a - b)$ er delelig med 3 **then**
 - 4.1. Output "de to strengene er i relasjon med hverandre"
- else**
 - 4.2. Output "de to strengene er ikke i relasjon med hverandre"

- Her kan både punkt 2, 3 og 4 gjøres mer detaljert.
- Det klarer dere på egenhånd.

Eksamen 12/6-06 Oppgave 4

La m og n være naturlige tall der $2 \leq m \leq n \leq 3$, og la V være en mengde med $m + n$ elementer, la oss si $V = \{A_1, \dots, A_m, B_1, \dots, B_n\}$. Vi kan da danne en enkel graf $K_{m,n}$ med V som nodemengde ved å si at det finnes nøyaktig en kant mellom A_i og B_j for hver $1 \leq i \leq m$ og hver $1 \leq j \leq n$, og ingen kanter ellers.

- (a) Lag en skisse av $K_{2,2}$, $K_{2,3}$ og $K_{3,3}$. Angi matrisen til $K_{2,3}$ (med hensyn på listingen A_1, A_2, B_1, B_2, B_3 av nodene).
- (b) Nøyaktig en av grafene fra a) er semi-Eulersk. Angi en Eulersti i denne grafen. En annen blant disse tre grafene er hverken Eulersk eller semi-Eulersk. Angi et utspennende tre for denne.

[Denne løser vi på tavla/overhead.]

Eksamen 12/6-06 Oppgave 5

For $n \in \mathbb{N}$, la $P(n)$ være følgende påstand:

$$\forall (a, b) \in \mathbb{Z} \times \mathbb{Z} \exists m \in \mathbb{Z} (a^n - b^n = m(a - b))$$

Vis at $P(n)$ er sann for alle $n \in \mathbb{N}$ ved induksjon.

(Hint: $a^{k+1} - b^{k+1} = a^{k+1} - ab^k + ab^k - b^{k+1}$)

Løsning

- For induksjonstarten, viser vi at $P(1)$ er sann.
- La $a, b \in \mathbb{Z}$.
- Det er tilstrekkelig å vise at det fins en m slik at $a^n - b^n = m(a - b)$.
- Ved å sette inn for n får vi $a^1 - b^1$.
- Ved å la $m = 1$ får vi at dette er lik $m(a - b)$.
- For induksjonsskrittet, så antar vi at $P(k)$ er sann for $k \in \mathbb{N}$.
- Vi må vise at $P(k + 1)$ er sann.
- La $a, b \in \mathbb{Z}$.
- Siden $P(k)$ er sann, fins det en m_1 slik at $a^k - b^k = m_1(a - b)$.
- Det er tilstrekkelig å vise at det fins en m slik at $a^{k+1} - b^{k+1} = m(a - b)$.

$$\begin{aligned} a^{k+1} - b^{k+1} &= a^{k+1} - ab^k + ab^k - b^{k+1} \text{ (dette er hintet)} \\ &= a(a^k - b^k) + b^k(a - b) \\ &= am_1(a - b) + b^k(a - b) \text{ (siden } P(k) \text{ er sann)} \\ &= (am_1 + b^k)(a - b) \end{aligned}$$
- Ved å la $m = am_1 + b^k$ har vi vist induksjonsskrittet.
- Ved induksjon følger det at $P(n)$ er sann for alle $n \in \mathbb{N}$.

Eksamen 31/5-2005 Oppgave 1

For hvert naturlig tall n la

$$t(n) = \binom{2n}{n} = \frac{(2n)!}{n!n!}$$

(a) Beregn $t(n)$ for $n = 1, 2$ og 3 .

(b) Vis at

$$t(n+1) = \frac{4n+2}{n+1}t(n)$$

for alle $n \geq 1$.

(c) Gi et induksjonsbevis for at

$$t(n) \leq 4^n$$

for alle $n \geq 1$. Hint: Bruk at $4n+2 \leq 4(n+1)$. Forklar både starten på induksjonen og induksjonsskrittet.

a) Beregn $t(n)$ for $n = 1, 2$ og 3 .

$$t(1) = \binom{2}{1} = \frac{2}{1} = 2$$

$$t(2) = \binom{4}{2} = \frac{4 \cdot 3}{2} = 6$$

$$t(3) = \binom{6}{3} = \frac{6 \cdot 5 \cdot 4}{3 \cdot 2} = 20$$

b)

$$\text{Vis at } t(n+1) = \frac{4n+2}{n+1} \cdot t(n) \text{ for alle } n \geq 1$$

Vi merker oss at $t(n+1) = \frac{(2(n+1))!}{(n+1)!(n+1)!}$ per definisjon.

$$\frac{4n+2}{n+1} \cdot t(n) = \frac{4n+2}{n+1} \cdot \frac{(2n)!}{n!n!} = \frac{(4n+2)(2n)!}{(n+1)n!n!} = \frac{2(2n+1)(2n)!}{(n+1)n!n!} =$$

$$\frac{2(2n+1)!}{(n+1)!n!} = \frac{2(2n+1)!(n+1)}{(n+1)!n!(n+1)} = \frac{(2n+2)(2n+1)!}{(n+1)!(n+1)!} =$$

$$\frac{(2n+2)!}{(n+1)!(n+1)!} = \frac{(2(n+1))!}{(n+1)!(n+1)!}$$

c) Gi et induksjonsbevis for at $t(n) \leq 4^n$ for alle $n \geq 1$. Hint: Bruk at $4n+2 \leq 4(n+1)$. Forklar både starten på induksjonen og induksjonsskrittet.

- For induksjonstarten, viser vi at påstanden er sann for $n = 1$:

$$- t(1) = 2, \text{ fra a), og } 2 \leq 4^1 = 4.$$

- For induksjonsskrittet, antar vi at påstanden holder for k .

$$- \text{Det vil si at } t(k) \leq 4^k.$$

- Vi må vise at påstanden holder for $k+1$.

$$- \text{Det vil si at } t(k+1) \leq 4^{k+1}.$$

- Vi regner på følgende måte:

$$t(k+1) = \frac{4n+2}{n+1} \cdot t(k) \leq \frac{4n+2}{n+1} \cdot 4^k \leq \frac{4(n+1)}{n+1} \cdot 4^k \leq 4 \cdot 4^k = 4^{k+1}$$

Eksamen 31/5-2005 Oppgave 2

En sammenhengende graf har 5 noder og 8 kanter. De fire første nodene har grad 1, 2, 3 og 4.

- (a) Hva er graden til det femte hjørnet?
- Siden antallet kanter er 8, må summen til gradene til nodene være 16.
 - Summen til gradene til de fire nodene som er angitt er $1 + 2 + 3 + 4 = 10$.
 - Da må den siste noden ha grad 6.
- (b) Er denne grafen et tre?
- Nei, for i ethvert tre er antall kanter lik antall noder minus 1.
- (c) Finnes det en Eulervei i denne grafen?
- Vi har bevist et teorem som sier at det fins en Eulersti hvis det er nøyaktig to noder av odde grad i grafen.
 - Siden kun nodene med grad 1 og 3 har odde grad, så fins det en Eulersti i grafen.

Eksamen 31/5-2005 Oppgave 3

En enkel vektet graf G med noder A, B, C, D, E og F har følgende vektmatrise.

	A	B	C	D	E	F
A	0	66	∞	38	∞	11
B	66	0	73	∞	13	∞
C	∞	73	0	42	∞	20
D	38	∞	42	0	91	∞
E	∞	13	∞	91	0	93
F	11	∞	20	∞	93	0

For eksempel er det en kant AB mellom A og B , med vekt 66, mens det ikke er noen kant i G mellom A og C .

- (a) Tegn et bilde av denne grafen, der hjørnene er merket A til F og hver kant $AB, AD, \text{etc.}$, er merket med den tilhørende vekten.
Hint: Noen av kantene må krysse hverandre i dette bildet. Tegn opp grafen en gang til for å besvare 3(b).
- (b) Bruk Prims algoritme til å finne et minimalt utspennende tre T for G . Svaret skal være en liste av de kantene i G som er med i T , der f.eks. kanten mellom A og B heter AB .

[Denne løser vi på tavla/overhead.]

Siste plenumsregning

Det var siste plenumsregning.

Vi avslutter med diskusjon og eventuelle spørsmål!

Lykke til på eksamen!

