Kræsjkurs MNF130

Steinar Simonnes og Carina Seidel

Institutt for Informatikk
Universitetet i Bergen

22. Mai 2024

Agenda

- Logikk
  - Proposisjoner
  - Kvantorer
- Bevis
  - Induksjon
  - Rekursjon
  - Strukturell induksjon
- Mengdelære
  - Mengder
  - Funksjoner
  - Tavleoppgaver
- Følger og summer
- Tallteori
- Stokastisitet
  - Counting
  - Sannsynligheter
- Algoritmer
  - Kompleksitet
  - Søking
- Grafer
  - Korteste sti

Slides 'n' Slido

Dere kan stille spørsmål digitalt og anonymt her:
`sli.do`, med koden "MNF130"

Dere finner presentasjonen og kildekoden på mittuib, eller her:
`tinyurl.com/MNF130-Slides`

Proposisjoner

- En proposisjon er et uttrykk med en sannhetsverdi, som alltid er enten Sann (T) elller Usann (F).
- Ofte setter vi dem i variabler, for å gjøre det lettere å lese.

Proposisjoner

p := $2 < 3$
q := "Månen er laget av ost"
r := "MNF130 er et nyttig fag for informatikere"

Ikke proposisjoner

"Hva skal vi ha til middag i dag?"
$x + y < z$

$\neg$, og sannhetstabeller

- Gitt en proposisjon p, kan vi representere den motsatte proposisjonen som $\neg p$.
- Vi kan tegne opp alle mulige verdier et slikt uttrykk kan ha i en tabell.

Eksempler

La $p := 2 < 3$.
$p =$ "Det er sant at 2 er mindre enn 3"
$\neg p =$ "Det er ikke sant at 2 er mindre enn 3"

p	$\neg p$
T	F
F	T

$\vee$ og $\wedge$

- Vi kan slå sammen flere proposisjoner til et større uttrykk på mange forskjellige måter.

Eksempel

La $p :=$ Jorden er flat, og $q :=$ Månen er flat
$p \vee q =$ Jorden er flat **eller** månen er flat
$p \wedge q =$ Jorden er flat **og** månen er flat

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

$\rightarrow$ og $\leftrightarrow$

- En proposisjon kan implisere en annen proposisjon.

Eksempel

La $p :=$ "Det regner ute", og $q :=$ "Bakken er våt"
$p \rightarrow q =$ "Hvis det regner ute, blir bakken våt"
$p \leftrightarrow q =$ "Det regner ute hvis og bare hvis bakken er våt"

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

Ekvivalenser og tautologier

- Når to logiske uttrykk alltid har samme verdi, sier vi at de er *logisk ekvivalente*.
- Om et uttrykk alltid er sant, uavhengig av innholdet, kaller vi det en tautologi.

p	$\neg p$	$\neg\neg p$	$p \vee \neg p$
T	F	T	T
F	T	F	T

- Konklusjon 1: $p \equiv \neg\neg p$
- Konklusjon 2: $p \vee \neg p \equiv T$, og er en tautologi
- For å se om to uttrykk er ekvivalente, tegn sannhetstabellen og se om kollonene deres er det samme.
- For å se om et uttrykk er en tautologi, tegn sannhetstabellen og se om kolonnen alltid er T.

Noen viktige logiske ekvivalenser

Ekvivalens	Navn
$p \wedge T \equiv p$ $p \vee F \equiv p$	Identity
$p \vee T \equiv T$ $p \wedge F \equiv F$	Domination
$p \vee p \equiv p$ $p \wedge p \equiv p$	Idempotent
$p \equiv \neg\neg p$	Negation
$p \vee q \equiv q \vee p$ $p \wedge q \equiv q \wedge p$	Commutative

Ekvivalens	Navn
$(p \vee q) \vee r \equiv p \vee (q \vee r)$ $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	Associative
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	Distributive
$\neg(p \wedge q) \equiv \neg p \vee \neg q$ $\neg(p \vee q) \equiv \neg p \wedge \neg q$	De Morgan
$p \vee (p \wedge q) \equiv p$ $p \wedge (p \vee q) \equiv q$	Absorption
$p \vee \neg p \equiv T$ $p \wedge \neg p \equiv F$	Negation

Flere viktige logiske ekvivalenser

Ekvivalenser med $\rightarrow$
$p \rightarrow q \equiv \neg p \vee q$
$p \rightarrow q \equiv \neg p \rightarrow \neg q$
$p \vee q \equiv \neg p \rightarrow q$
$p \wedge q \equiv \neg(p \rightarrow \neg q)$
$\neg(p \rightarrow q) \equiv p \wedge \neg q$
$(p \rightarrow q) \wedge (p \rightarrow r) \equiv p \rightarrow (q \wedge r)$
$(p \rightarrow q) \wedge (q \rightarrow r) \equiv (p \vee q) \rightarrow r$
$(p \rightarrow q) \vee (p \rightarrow r) \equiv p \rightarrow (q \vee v)$
$(p \rightarrow r) \vee (q \rightarrow r) \equiv (p \wedge q) \rightarrow r$

Ekvivalenser med $\leftrightarrow$
$p \leftrightarrow q \equiv p \rightarrow q \wedge q \rightarrow p$
$p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$
$p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$
$\neg(p \leftrightarrow q) \equiv p \rightarrow \neg q$

Typisk logikkoppgave

Vis at $(p \wedge \neg q) \to \neg r$ og $(p \wedge r) \to q$ er logisk ekvivalente, ved å bruke enkle logiske ekvivalenser.

$$
\begin{aligned}
&(p \wedge \neg q) \to \neg r \\
&\equiv \neg(p \wedge \neg q) \vee \neg r \\
&\equiv (\neg p \vee \neg\neg q) \vee \neg r \\
&\equiv (\neg p \vee q) \vee \neg r \\
&\equiv \neg p \vee q \vee \neg r \\
&\equiv \neg p \vee \neg r \vee q \\
&\equiv \neg(p \wedge r) \vee q \\
&\equiv (p \wedge r) \to q \qquad \square
\end{aligned}
$$

Predikater

- Et predikat er bare en funksjon som returnerer T eller F.
- Gitt et predikat og riktig antall argumenter, kan vi evaluere det som en vanlig proposisjon.

Eksempler på predikater

$P(x, y, z) = x + y < z$
$Q(s) = s$ contains $'a'$

Eksempler på evaluering

$P(1, 2, 3) = 1 + 2 < 3 = 3 < 3 = F$
$Q($«Steinar»$) =$«Steinar» contains $'a' = T$

Kvantorer (/Quantifiers)

- Ofte vil vi si noe om et predikat $P(x)$, men for flere potensielle x samtidig.
- $\forall x : P(x) = x_1 \wedge x_2 \wedge ... \wedge x_n =$ "For alle x er det sant at $P(x)$"
- $\exists x : P(x) = x_1 \vee x_2 \vee ... \vee x_n =$ "Det eksisterer en x slik at det er sant at $P(x)$"
- $\exists x \in S : P(x) =$ "Det eksisterer en x i settet S slik at det er sant at $P(x)$"

Eksempler

$\forall x : (x < x + 1) =$ "For alle x er det sant at $x < x + 1$"
$\exists x : (x > x^2) =$ "Det eksisterer en x slik at det er sant at $x > x^2$"
$\exists x \in \mathbb{N} : (x * x = 1)$

Kvantorer i praksis

Kvantorer ser skumle ut, men det er bare for-løkker.

```
def forall(p, xs):
    for x in xs:
        if not p(x):
            return False
    return True

def exists(p, xs):
    for x in xs:
        if p(x):
            return True
    return False
```

- exists(p, xs) $= \exists x \in xs : p(x)$
- forall(p, xs) $= \forall x \in xs : p(x)$

Nøstede kvantorer (/Nested quantifiers)

- Ofte vil vi si noe om mange kombinasjoner av variabler samtidig.
- Da slår vi sammen flere $\forall$ og $\exists$.
- Om vi har flere like kvantorer etter hverandre kan vi slå dem sammen med tupler.

Eksempler

$\forall x \forall y : (x^2 + y^2 \geq 0) = \forall x, y : (x^2 + y^2 \geq 0)$
$\forall x \exists y : (x * y = 1) =$ "For alle x eksisterer det en y slik at $x * y = 1$"
$\exists x \forall y : (x^2 \leq y^2) =$ "Det eksisterer en x slik at for alle y er $x^2 \leq y^2$"
$\neg \exists n, a, b, c \in \mathbb{Z} : (n > 2 \wedge a^n + b^n = c^n)$

De Morgan dukker opp igjen

Det viser seg at De Morgan's lover også fungerer på kvantorer.

- $\neg\forall x : P(x)$
  $\equiv \neg[P(x_1) \wedge P(x_2) \wedge ... \wedge P(x_n)]$
  $\equiv \neg P(x_1) \vee \neg P(x_2) \vee ... \vee \neg P(x_n)$
  $\equiv \exists x : \neg P(x)$
- $\neg\exists x : P(x)$
  $\equiv \neg[P(x_1) \vee P(x_2) \vee ... \vee ..P(x_n)]$
  $\equiv \neg P(x_1) \wedge \neg P(x_2) \wedge ... \wedge \neg P(x_n)$
  $\equiv \forall x : \neg P(x)$

∀ 🐠 ∃ 🐟 (🐟 > 🐠)

Spørsmål?

Figur: Guillaume på Vidden

Beviser

Gitt noen premisser, vis at det garanterer en konklusjon.

Det aller vanskeligste er å finne riktig idé til beviset. Det finnes mange typer bevis, og det er mye rom for kreativitet. Man må bare prøve seg frem.

Likvel kan vi vanligvis begynne med følgende:

1. Vurder hvorvidt påstanden ser ut til å stemme. Test på noen eksempler, prøv å finne et moteksempel.
2. Skriv opp hva vi vet, og hva vi vil vise, i mattenotasjon
3. Se etter en sammenheng, grunnen til at det ene impliserer det andre.
4. Gjør resten av beviset.

Oppgave: vis at om a og b er oddetall, da er $a + b$ et partall.

1. Vi tester med et par eksempler: $3 + 7 = 10, 15 + 1 = 16$.
2. Vi vet at $a = 2k + 1$, og $b = 2c + 1$, for noen heltall k, c.
   Vi vil vise at $a + b = 2h$, for et heltall h.
3. Vi prøver å bare legge dem sammen og se hva som skjer:
   $a + b = (2k + 1) + (2c + 1) = 2(k + c + 1)$
   Og da var vi plutselig allerede ferdige.

Bevis: summen av to oddetall er et partall

La $a = 2k + 1, b = 2c + 1$ være oddetall, for vilkårlige heltall k og c.
Da er $a + b = (2k + 1) + (2c + 1) = 2(k + c + 1)$, som er et partall. $\square$

Direkte og kontrapositive bevis

Et direkte bevis for $p \to q$ er et som antar at $p = \mathsf{T}$, og viser at det medfører at $q = \mathsf{T}$. Dette er ofte de enkleste bevisene. Hvis du har formler du bare må knytte sammen, forsøk det først.

Vi gjorde nettopp et direkte bevis: $p =$ "a og b er heltall", $q =$ "$a + b$ er et oddetall", og vi viste at $p \to q$.

Siden $p \to q \equiv \neg q \to \neg p$, kan vi også bevise det 'baklengs': om konklusjon er usann, må premisset også være usannt.

Oppgave: vis at for alle n, om n^2 er et partall, er n et partall.

1. Vi tester: 64 og 8 er partall, 36 og 6 er partall. Ser riktig ut.
2. Vi vet at $n^2 = 2k$, og vil vise at da er $n = 2c$.
3. Om n^2 er et partall, må det antageligvis være et partall ganget med seg selv, ellers får vi et oddetall. Det høres meningsfullt ut, men er kanskje vanskelig å vise rett frem.

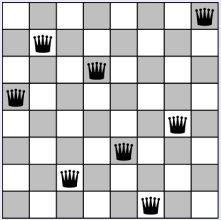
For alle n, om n^2 er et partall, er n et partall

La $n = 2c + 1$ være et oddetall, for et heltall k.
Da er $n^2 = (2c+1)^2 = 4c^2 + 4c + 1 = 2(2c^2 + 2c) + 1$, som er et oddetall.
Om n er et oddetall er n^2 dermed også det. Om n^2 er et partall, er n dermed nødt til å være et partall. $\square$

Motbevis

For å motbevise en påstand, holder det å bare finne et moteksempel. Dette er vanligvis den letteste typen bevis.
Det er kun her det er greit å stoppe etter kun et eksempel.

Påstand: det er ikke mulig å plassere 8 dronninger på et sjakkbrett uten at noen av dem truer hverandre.



Motsigelsesbevis (/proof by contradiction)

For å bevise en proposisjon p, kan vi heller bevise at $\neg p$ leder til en selvmotsigelse. Vi motbeviser det motsatte.

Et motsigelsesbevis ser vanligvis slik ut:

Vis at p

Ok da, la oss anta at p på en eller annen måte *ikke* er sant.
Hadde ikke det vært ganske dumt?
Ja nettopp, jeg regner med du føler deg dum nå.
Derfor er p sant.

Motsigelsesbevis

Summen av et rasjonalt tall $\frac{a}{b}$ og et irrasjonalt tall c også er et irrasjonalt tall.

Vi antar det motsatte, at summen blir et rasjonalt tall:

$\frac{a}{b} + c = \frac{e}{d}$ for $e, d \in \mathbb{Z}$.

$\implies \frac{e}{d} - \frac{a}{b} = c$

$\implies \frac{b \cdot d - a \cdot d}{b \cdot d} = c$

Men det impliserer at c er et rasjonalt tall. ↯

Derfor er det usant at summen er et rasjonalt tall

$\implies$ summen må være irrasjonal $\square$

Uttømmende bevis (/proof by exhaustion)

Noen ganger greier vi ikke finne på et elegant bevis. Da tar vi heller for oss hvert enkelt tilfelle hver for seg, og når vi har vist at det holder for absolutt alle tilfeller har vi bevist påstanden. Merk at da må det være et endelig antall tilfeller.

Vis at alle sommer-OL har blitt arrangert i årstall delelige på 4.

$1896 \bmod 4 = 0$ ✓
$1900 \bmod 4 = 0$ ✓
[... de neste 30 linjene er trivielle og etterlatt som en oppgave for leseren ...]
$2024 \bmod 4 = 0$ ✓

Dette er altså ikke et komplett bevis. Vi må faktisk vise alle tilfeller før vi er ferdig.

Matematisk induksjon

Et induksjonsbevis for en påstand $P(x)$ består av to steg:

- Vis at $P(0)$ stemmer
- Vis at om $P(k)$ stemmer, må $P(k+1)$ stemme

Siden $P(0)$ stemmer, må $P(1)$ stemme, og
Siden $P(1)$ stemmer, må $P(2)$ stemme, og
...
Siden $P(k)$ stemmer, må $P(k+1)$ stemme.

Dermed har vi bevist at $\forall x : x \geq 0 \rightarrow P(x)$.

Eksempel på matematisk induksjon

Vis at $\sum_{i=0}^{n} i = 0 + 1 + 2 + ... + n = \frac{n(n+1)}{2}$.

Vi lar $P(k) := \sum_{i=0}^{k} i = \frac{k(k+1)}{2}$

Vi begynner med base case, at påstanden holder for $P(0)$:
$P(0) : \sum_{i=0}^{0} i = \frac{0(0+1)}{2} = 0$ ✓

Eksempel på matematisk induksjon (fortsettelse)

Vis at om $P(k)$ stemmer, må $P(k+1)$ stemme.

Induksjonshypotesen vår er at påstanden holder for k:
$P(k) : \sum_{i=0}^{k} i = \frac{k(k+1)}{2}$

Nå vil vi vise at det medfører at påstanden også må holde for $k+1$:
$P(k+1) : \sum_{i=0}^{k+1} i = \frac{(k+1)((k+1)+1)}{2} = \frac{(k+1)(k+2)}{2}$

$\sum_{i=0}^{k+1} i = 0 + 1 + 2 + ... + k + (k+1) = (\sum_{i=0}^{k} i) + (k+1)$
$= \frac{k(k+1)}{2} + k + 1$
$= \frac{k(k+1)}{2} + \frac{2(k+1)}{2} = \frac{k(k+1)+2(k+1)}{2}$
$= \frac{(k+1)(k+2)}{2}$ ✓

Sterk induksjon

Dette er veldig likt. Men istedet for å anta $P(k)$, antar man $P(0) \wedge P(1) \wedge .... \wedge P(k)$. Du kan tenke helt likt på disse to formene for induksjon, eneste forskjellen er at du kan anta litt mer i induksjonssteget.

Hvis du allerede forstår induksjon, trenger du ikke definisjon for sterk induksjon. Du kan gjørde det med intuisjon.

Rekursive funksjoner

En rekursiv funksjon er en funksjon som refererer til seg selv.

fakultet

$0! := 1$
$n! := n \cdot (n-1)!$

$$
\begin{aligned}
3! &= 3 \cdot \ 2! \\
&= 3 \cdot 2 \cdot 1! \\
&= 3 \cdot 2 \cdot 1 \cdot 0! \\
&= 3 \cdot 2 \cdot 1 \cdot 1 \\
&= 6
\end{aligned}
$$

Evaluering av en rekursiv funksjon

$$fib(0) := 0$$
$$fib(1) := 1$$
$$fib(n) := fib(n-1) + fib(n-2)$$

$$\begin{aligned}
&fib(5) = fib(3) + fib(4) \\
&= [fib(1) + fib(2)] + [fib(3) + fib(2)] \\
&= [1 + fib(0) + fib(1)] + [fib(2) + fib(1) + fib(1) + fib(0)] \\
&= [1 + 0 + 1] + [fib(1) + fib(0) + 1 + 1 + 0] \\
&= [1 + 0 + 1] + [1 + 0 + 1 + 1 + 0] \\
&= 5
\end{aligned}$$

Rekursive definisjoner

Som med funksjoner kan vi også definere andre strukturer rekursivt.

De defineres med et basissteg, dvs utgangspunktet, og et rekursivt steg for å utvide det.

Eksempel

Basissteg: $7 \in \mathbb{S}$.
Rekursivt steg: $a \in \mathbb{S} \rightarrow 10a \in \mathbb{S}$
$\mathbb{S} = \{7, 70, 700, 7000, 70000, ...\}$

Strukturell induksjon

Ofte vil vi bevise egenskaper for rekursive strukturer. Det gjøres ved to steg:

- Basissteget: vis at strukturen i utgangspunktet har egenskapen.
- Det rekursive steget: vis at om en struktur allerede har en egenskap, vil en runde med rekursjon opprettholde den egenskapen.

Vis at alle tall i $\mathbb{S}$ er delelige på 7.

Basisteget: $\mathbb{S}$ inneholder bare 7, og 7 mod 7 = 0. ✓
Rekursive steget: Vi antar at alle tall $a \in \mathbb{S}$ er delelig på 7. Så $a = 7k$, $k \in \mathbb{N}$.
I neste rekursjon legger vi til $10a$ i $\mathbb{S}$.
$10a = 10 \cdot 7k$, og er også delelig på 7. En runde rekursjon opprettholder egenskapen.✓
Derfor er alle tall i $\mathbb{S}$ delelige på 7.

Større eksempel på strukturell induksjon

Basissteg: $(0,0) \in \mathbb{T}$.
Rekursivt steg: $(a,b) \in \mathbb{T} \rightarrow (a+1,b) \in \mathbb{T} \wedge (a+1,b+1) \in \mathbb{T}$.

Oppgave: vis at $\forall a,b : [(a,b) \in \mathbb{T} \rightarrow a \geq b]$.

Basissteg: $(0,0) \in \mathbb{T}$, og $0 \geq 0$. ✓
Rerkursivt steg: vi antar at $\forall a,b : [(a,b) \in \mathbb{T} \rightarrow a \geq b]$. Hvert rekursive kall legger til to nye par, $(a+1,b)$ og $(a+1,b+1)$. Vi ser på dem hver for seg:

- Om $a \geq b$ er $a+1 \geq b$.
- Om $a \geq b$ er $a+1 \geq b+1$. ✓

Dermed kan vi konkludere med at $\forall a,b : [(a,b) \in \mathbb{T} \rightarrow a \geq b]$.

Strukturell induksjon i praksis

```
public class SimpleList<T> {
    private final T[] content;
    private int size = 0;
    public SimpleList(int capacity) {
        content = (T[]) new Object[capacity];
    }
    public void add(T element) {
        content[size] = element;
        size++;
    }
    public T get(int index) { return content[index]; }
    public int size() { return size; }
}
```

Spørsmål?

Figur: Guillaume på Blåmanen

Sett / Mengder

Et sett, eller en mengde, er en slags liste av objekter, bortsett fra:

1. Det inneholder ingen duplikater
2. Det har ingen konkret rekkefølge

Eksempel

Alle de følgende settene er like:
$\{1, 2, 3\} = \{3, 2, 1\} = \{1, 1, 1, 1, 2, 2, 3, 3\}$

Eksempel

$1 \in \{1, 2, 3\} = \mathsf{T}$
$5 \in \{1, 2, 3\} = \mathsf{F}$

Kjente sett

Flere sett bruker vi veldig ofte, og har fått designerte symboler. Her er noen av dem.

Symbol	Navn	Innhold
$\mathbb{N}_0$	Naturlige tall	$\{0, 1, 2, 3, ...\}$
$\mathbb{N}_1$	Naturlige tall	$\{1, 2, 3, ...\}$
$\mathbb{N}$	Naturlige tall	Enten $\mathbb{N}_0$ eller $\mathbb{N}_1$
$\mathbb{Z}$	Heltall	$\{... - 2, -1, 0, 1, 2, 3, ...\}$
$\mathbb{Q}$	Rasjonale tall	$\{-3/2, 1/10, 4/5, 5/4, ...\}$
$\mathbb{R}$	Reelle tall	$\{\pi, e, 0.111111111..., -2\pi, ...\}$
$\emptyset$	Det tomme settet	$\{\}$

Settbyggingsnotasjon

Istedet for å manuelt skrive opp alle elementene i et sett, eller å bruke uformell «...»-notasjon, kan vi bruke setbyggingsnotasjon.

Eksempler

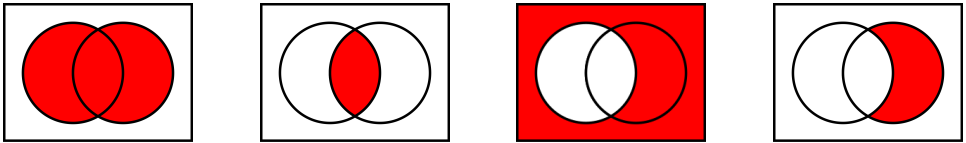
$\{x \mid x \in \mathbb{N}\} = \mathbb{N}$
$\{x \cdot 2 \mid x \in \mathbb{N}\} = \{0, 2, 4, 6, 8, ...\}$
$\{x \mid x \in \mathbb{N},\ x \text{ mod } 3 = 0\} = \{0, 3, 6, 9, ...\}$
$\{x/2 \mid x \in \mathbb{N},\ x \text{ mod } 3 = 0\} = \{0, 1.5, 3, 4.5, ...\}$

Det ser skummelt ut, men det er bare forløkker.

```
[ x for x in range(100) ]
[ x*2 for x in range(100) ]
[ x for x in range(100) if x % 3 == 0 ]
[ x/2 for x in range(100) if x % 3 == 0 ]
```

$\cup$, $\cap$, $\bar{}$ og $-$

$$A \cup B := \{x \mid x \in A \vee x \in B\}$$
$$A \cap B := \{x \mid x \in A \wedge x \in B\}$$
$$A^C = \bar{A} := \{x \mid x \notin A\}$$
$$A - B = A \backslash B := \{x \mid x \in A \wedge x \notin B\} = A \cap \bar{B}$$



(a) $A \cup B$ (b) $A \cap B$ (c) $\bar{A}$ (d) $B - A$

$\subset, \subseteq, =, \supseteq, \supset$

Vi har flere måter å uttrykke at et sett inneholder elementer fra et annet sett.

- $A \subseteq B := \forall x \in A : (x \in B)$
- $A \subset B := A \subseteq B \wedge \exists x \in B : (x \notin A)$
- $A = B := A \subseteq B \wedge B \subseteq A$

Eksempler

$\{1\} \subseteq \{1, 2\} = \mathsf{T}$
$\{1\} \subseteq \{5\} = \mathsf{F}$
$\{a, b\} \subset \{a, b\} = \mathsf{F}$
$\{a, b\} \subseteq \{a, b\} = \mathsf{T}$

Obs! Enkelte skriver $\subset$ når de mener $\subseteq$, og $\subsetneq$ når de mener $\subset$.

Nyttige regler for sett

Ekvivalens	Navn
$A \cap U = A$ $A \cup \emptyset = A$	Identity
$A \cup U = U$ $A \cap \emptyset = \emptyset$	Domination
$A \cup A = A$ $A \cap A = A$	Idempotent
$A = (A^C)^C$	Negation
$A \cup B = B \cup A$ $A \cap B = B \cap A$	Commutative

Ekvivalens	Navn
$(A \cup B) \cup C = A \cup (B \cup C)$ $(A \cap B) \cap C = A \cap (B \cap C)$	Associative
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	Distributive
$(A \cap B)^C = A^C \cup B^C$ $(A \cup B)^C = A^C \cap B^C$	De Morgan
$A \cup (A \cap B) = A$ $A \cap (A \cup B) = A$	Absorption
$A \cup A^C = U$ $A \cap A^C = \emptyset$	Negation

Tavleoppgaver fra H19

- Vis eller motbevis at $(A - C) \cap (B - C) = \emptyset$.
- Vis eller motbevis at $(A - C) \cap (C - B) = \emptyset$.

For å løse det med vennediagrammer:

1. Tegn et vennediagram. Begynn med sirkler for A, B, etc.
2. Fargelegg områdene til deluttrykkene, dvs $(A - C)$ og $(B - C)$ i dette eksemplet.
3. Bruk områdene i deluttrykkene til å farge omårdene i de større uttrykkene, dvs hele $(A - C) \cap (B - C)$ her.
4. Er områdene til hele uttrykkene på hver side av = det samme?

Alternativt: bruk regnereglene for $\cap$, $\cup$, etc til å omformulere uttrykket.

Tricky oppgave

Vi har to sett A og B, og vi får vite at $A - B = \emptyset$. Hva vet vi da om A og B?

- $A = \emptyset$
- $B = \emptyset$
- $A \subset B$
- $A \subseteq B$
- $B \subset A$
- $B \subseteq A$
- $A \cap B \neq \emptyset$
- $A \neq B$
- $A = B$
- $B - A \neq \emptyset$
- $B - A = \emptyset$
- $A \cup B = \emptyset$

Kardinalitet

Om A er et sett, er $|A|$ antall elementer i settet, *kardinaliteten*, eller *lengden*.
To sett har samme kardinalitet om de har samme lengde.

Eksempler

$|\{a, b, c\}| = 3$
$|\{\}| = |\emptyset| = 0$
$|\mathbb{N}| = \aleph_0$

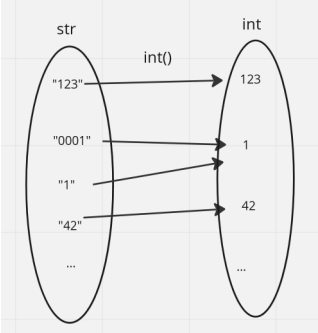
Funksjoner

Funksjoner kjenner vi fra INF100 i fjor. Nå skal vi formalisere dem litt mer.

Fra nå av ser vi på datatyper som sett:

- $int := \mathbb{Z}$
- $str :=$ { "a", b", "c", "aa", "ab", ... }

Domenet til en funksjon er inputsettet.
Kodomenet til en funksjon er outputsettet.



Figur: $int : str \to int$

Funksjonskomposisjon

Gitt to funksjoner:

- $f : A \to B$
- $g : B \to C$

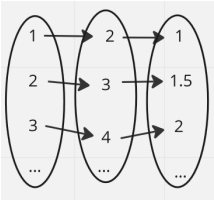
Kan vi definere en unik tredje funksjon:

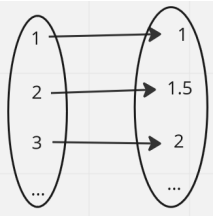
$g \circ f : A \to C$

$g \circ f := g(f(x))$

La $f(x) := x + 1$, og $g(x) := x/2$.

Da er $g \circ f = g(f(x)) = (x + 1)/2$.

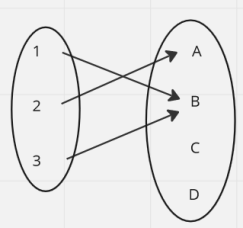




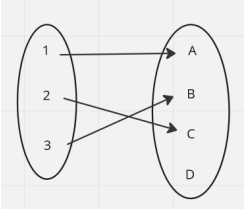
Obs! Rekkefølgen er uintuitiv. f skjer før g. Dere kan lese det som g 'anvendt på' f.

Injektivitet

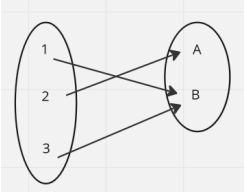
En funksjon $f : A \to B$ er *injektiv* hvis $\forall a_1, a_2 \in A : [f(a_1) = f(a_2) \to a_1 = a_2]$.
Med andre ord: alle inputs gir et unikt output og alle piler peker på forskjellige ting.



(a) Ikke injektiv



(b) Injektiv

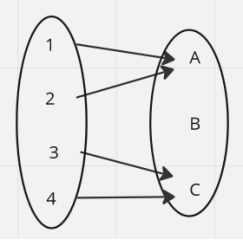


(c) Ikke injektiv

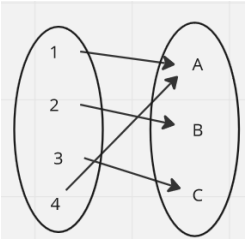
Observer at det finnes en injeksjon hvis og bare hvis $|B| \geq |A|$.

Surjektivitet

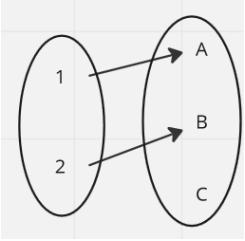
En funksjon $f : A \to B$ er *surjektiv* hvis $\forall b \in B\ \exists a \in A : [f(a) = b]$.
Med andre ord: hele kodomenet/outputsettet blir brukt og alt i B har minst én pil som peker på seg.



(a) Ikke surjektiv



(b) Surjektiv

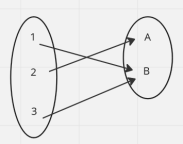


(c) Ikke surjektiv

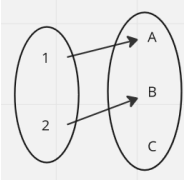
Observer at det finnes en surjeksjon hvis og bare hvis $|A| \geq |B|$.

Bijektivitet

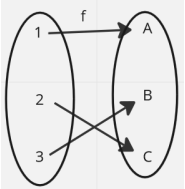
En funksjon $f : A \to B$ er *Bijektiv* om den er både injektiv og surjektiv.
Da finnes det en unik invers funksjon $f^{-1} : B \to A$ slik at $f^{-1} \circ f = id_A$ og $f \circ f^{-1} = id_B$, dvs at $\forall a \in A : [a = f^{-1}(f(a))]$ og $\forall b \in B : [b = f(f^{-1}(b))]$.



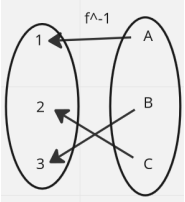
(a) Surjektiv, ikke injektiv



(b) Injektiv, ikke surjektiv



(c) Bijektiv



(d) Inverset f^{-1}

Observer at det finnes en bijeksjon hvis og bare hvis $|A| = |B|$.

- Injektiv: $\forall a_1, a_2 : [f(a_1) = f(a_2) \rightarrow a_1 = a_2]$.
- Surjektiv: $\forall b \in B\ \exists a \in A : [f(a) = b]$.
- Bijektiv: begge de over

Avgjør om de følgende funksjonene er injektive, surjektive, eller bijektive:

$f : \mathbb{Z} \rightarrow \mathbb{R}$
$f(x) := x^2$

$even : \mathbb{Z} \rightarrow \text{Bool}$
$even(x) := x \text{ mod } 2 == 0$

$h : \mathbb{R} \rightarrow \mathbb{R}$
$h(x) := 2x + 1$

$g : \mathbb{R} \rightarrow \mathbb{R}$
$g(x) := e^x$

$h : \mathbb{R} \rightarrow \mathbb{R}^+$
$h(x) := e^x$

Noen kaller surjektive funksjoner for 'onto' på engelsk. Mens 'one-to-one', eller 1-1, kan bety enten bijektiv eller injektiv avhengig av hvem som sier det.

Spørsmål?

Figur: Guillaume på Vidden

Følger (Sequences)

Definisjon

- diskret struktur som representerer en ordnet liste
- funksjon fra en undermengde av heltall (vanligvis $\mathbb{N}$ eller $\mathbb{N}_0$) til en mengde S
- vanlig notasjon er $\{a_n\}$ der a_n kalles for en term av følgen.
- OBS! IKKE bland med mengdenotasjon!

Eksempel: Følge $\{a_n\}$ der $a_n = \frac{1}{n}$, altså $a_1 = \frac{1}{1} = 1, a_2 = \frac{1}{2}, \ldots$

Geometrisk Progresjon (Geometric Progression)

Definisjon

- Følge som har form $a, ar, ar^2, ..., ar^n, ...$
- a kalles for *startterm* (initial term) og r kalles for *fellesforhold* (common ratio)

Eksempler:

- Følge $\{b_n\}$ der $b_n = 2 \cdot 5^n$, altså $b_1 = 2 \cdot 5^0 = 2, b_2 = 10, b_3 = 50, ...$
- Følge $\{c_n\}$ der $c_n = 6 \cdot \left(\frac{1}{3}\right)^n$, altså $c_1 = 6 \cdot \left(\frac{1}{3}\right)^0 = 6, c_2 = 2, c_3 = \frac{2}{3}, ...$

Aritmetisk Progresjon (Arithmetic Progression)

Definisjon

- følge som har form $a, a+d, a+2d, ..., a+nd, ...$
- a kalles for *startterm* og d kalles for *fellesdifferanse* (common difference)

Eksempler:

- Følge $\{s_n\}$ der $s_n = -1 + 4n$, altså $s_1 = -1 + 4 \cdot 0 = -1, s_2 = 3, s_3 = 7, ...$
- Følge $\{t_n\}$ der $t_n = 7 - 3n$, altså $t_1 = 7 - 3 \cdot 0 = 7, t_2 = 4, t_3 = 1, ...$

Rekurrensrelasjoner (Recurrence Relations)

Definisjon

- uttrykker a_n med forrige termer i følgen, dvs. noen av $a_0, a_1, ..., a_{n-1}$
- en følge kalles for løsning av en rekurrensrelasjon hvis det tilfredstiller alle kravene.

Eksempler:

- La $\{a_n\}$ være løsningen for rekurrensrelasjonen $a_n = a_{n-1} + 3$ for $n \in \mathbb{N}$ og $a_0 = 2$. Hva er a_1, a_2 og a_3?
- Svar: $a_1 = 5$, $a_2 = 8$ og $a_3 = 11$, ikke noe overraskelse her
- Fun fact: Kravet $a_0 = 2$ kalles for *startbetingelsen* (initial condition)

Summeringer (Summations)

$$a_m + a_{m+1}? \ldots + a_n = \sum_{j=m}^{n} a_j$$

j heter da *summeringsindeks*, m heter *nedre grense* og n heter *øvre grense*

Geometrisk rekke (Geometric series)

For $a \in \mathbb{R}$ og $r \in \mathbb{R} \setminus \{0\}$

$$\sum_{j=0}^{n} ar^j = \begin{cases} \frac{ar^{n+1}-a}{r-1} & \text{if } r \neq 0 \\ (n+1)a & \text{if } r = 0 \end{cases}$$

Divisjon og Modulær aritmetikk

Delelighet $a|b$ (a deler b)

a kan dele b uten rest
$a|b$ er det samme som $\frac{b}{a} = c$ eller $b = a \cdot c$ med $c \in \mathbb{Z}$
Eksempel: $3|12$ eller $\frac{12}{3} = 4$ eller $12 = 3 \cdot 4$

Modulo (Klokkearitmetikk)

$a \, mod \, b$ gir ut resten av heltall divisjon av $\frac{a}{b}$ ($a\%b$ i programmeringsspråk) $a \, mod \, b = r$ kalles *remainder* Eksempel: $17 \, mod \, 5 = 2$ fordi $17 = 3 \cdot 5 + 2$

Algoritme for divisjon /modulo

- $d = q \cdot a + r$ med
- $q = \left\lfloor \frac{d}{a} \right\rfloor$ og $r = d \, mod \, a$
- Eksempel: $q = \left\lfloor \frac{17}{5} \right\rfloor = \lfloor 3.4 \rfloor = 3$
- $17 = 3 \cdot 5 + r \iff 17 = 15 + r \iff r = 2$

Modulo regneregler

Kongruens $\equiv$

- $a \equiv b \pmod{m}$: a og b kongruent i forhold til mod m
- $a \equiv b \pmod{m}$ betyr $a \mod m = b \mod m$
- vi skriver $[a]_m := a \pmod{m}$
- Eksempel: $8 \equiv 3 \pmod{5} \equiv [3]_5$ betyr $[8]_5 = 3 = [3]_5$

- Addisjon: $[a+b]_m = [[a]_m + [b]_m]_m$
- $[8+21]_6 = [[8]_6 + [21]_6]_6 = [2+3]_6 = [5]_6 = 5$
- Multiplikasjon: $[a \cdot b]_m = [[a]_m \cdot [b]_m]_m$
- $[8 \cdot 21]_6 = [[8]_6 \cdot [21]_6]_6 = [2 \cdot 3]_6 = [6]_6 = 0$

Eksempel

- $x \equiv 3\,(mod\,5)$ eller $[x]_5 = 3$
- $y \equiv 4\,(mod\,5)$ eller $[y]_5 = 4$
- Finn løsningen: $(3 \cdot x + 2 \cdot y^2)\,mod\,5$

$$[3 \cdot x + 2 \cdot y^2]_5 = [[3 \cdot x]_5 + [2 \cdot y^2]_5]_5$$

$$[3 \cdot x]_5 = [[3]_5 \cdot [x]_5]_5 = [3 \cdot 3]_5 = [9]_5 = 4$$
$$[2 \cdot y^2]_5 = [[2]_5 \cdot [y \cdot y]_5]_5 = [[2]_5 \cdot [y]_5 \cdot [y]_5]_5 = [2 \cdot 4 \cdot 4]_5 = [32]_5 = 2$$

$$[[3 \cdot x]_5 + [2 \cdot y^2]_5]_5 = [4 + 2]_5 = [6]_5 = 1$$

Modulo ved subtraksjon

Vi vet at vi har addisjon, men hva er med subtrasjon?

Substraksjon:
$[6-3]_8 = [3]_8$
$[3-6]_8$?
$[3-6]_8 = [-3]_8 = [0-3]_8 = [8-3]_8 = [5]_8 = 5$

Subtraksjon fungerer også for modulo.

Modulo ved divisjon

Vi vet at vi har multiplikasjon, men hva med divisjon?

Divisjon:
$[6/3]_8 = [2]_8$? ja, fordi $[2 \cdot 3]_8 = [6]_8$
$[3/6]_8$?
Nei, noen ganger fungerer det, noen ganger fungerer det ikke.

Vi kan ikke alltid dele!

Tallsystem

En representasjon av tall med forskjellige tegn med en base

Navn	Sifre	5	11	34
Desimal (b=10)	0-9	5	11	34
Binær (b=2)	0-1	101	1011	100010
Octal (b=8)	0-7	5	13	42
Hexadesimal (b=16)	0-9,a-f	5	B	22
base=13	0-9,a-c	5	B	28

Tabell: Eksempler på forskjellige tallsystemer

Desimal til base b

pseudokode

tall n til base b:
next digit $= n\%b$
$n = \lfloor \frac{n}{b} \rfloor$
forsette med det til $n = 0$

n	nextDigit	output
22		0
11	0	0
5	1	10
2	1	110
1	0	0110
0	1	10110

Tabell: Eksempel for dec til base 2

Base b til desimal

pseudokode

tall n og base b
$sum = 0$; $index = 0$
starter med først siffer s:
$sum+ = base^{index} \cdot s$
$index+ = 1$
forsette med hver siffer s

tall	1	0	1	1	0
base	16	8	4	2	1
produkt	16	0	4	2	0

Tabell: Eksempel for 2 til dec

$$16 + 0 + 4 + 2 + 0 = 22$$

GCD

La oss si at vi har en brøk: $\frac{12}{18}$. Hvordan forenkle den?
Vi deler telleren og nevneren på 6, og får $\frac{2}{3}$. Men hvorfor 6?
Fordi 6 er det største tallet som deler både 12 og 18. Den største felles faktoren er 6.

Største felles faktor (/Greatest Common Divisor)

$gcd(a, b) :=$ det største tallet som deler både a og b
Eksempel: $gcd(12, 18) = 6$.

Relativt primisk (/co-prime)

To heltall a og b kalles *relativt prime*, eller *co-prime*, dersom $gcd(a, b) = 1$.
Eksempel: 8 og 21 er relativt prime, siden $gcd(8, 21) = 1$.

LCM

Vi har en sum av to brøker: $\frac{5}{18} + \frac{7}{12}$. Hvordan forenkle det?
Vi ganger den venstre med 2, og den høyre med 3, så begge får 36 i nevneren:
$\frac{10}{36} + \frac{21}{36} = \frac{31}{36}$. Men hvorfor 36?
Fordi 36 er det laveste tallet som begge nevnerene kan ganges opp til, deres laveste felles multiplum er 36.

Laveste felles multiplum (/Least common multiple)

$lcm(a, b) :=$ det minste tallet som kan deles av både a og b
Eksempel: $lcm(18, 12) = 36$
Det er alltid sant at $a \cdot b = lcm(a, b) \cdot gcd(a, b)$, så vi kan regne ut LCM med $lcm(a, b) = \frac{a \cdot b}{gcd(a,b)}$.

Euklids algoritme

```
def gcd(a, b):
    while b != 0:
        r = a % b
        a = b
        b = r
    return a
```

$gcd(\underline{1180},\underline{482})$:

$$\underline{1180} = 2 \cdot \underline{482} + \underline{216}$$
$$\underline{482} = 2 \cdot \underline{216} + \underline{50}$$
$$\underline{216} = 4 \cdot \underline{50} + \underline{16}$$
$$\underline{50} = 3 \cdot \underline{16} + \underline{2}$$
$$\underline{16} = 8 \cdot \underline{2} + 0$$

$$gcd(1180, 482) = 2.$$

Utvidet Euklids algoritme

Regner ut to parameter s og t slik at $gcd(a, b)$ kan skrives som linærkombinasjon
$gcd(a, b) = s \cdot a + t \cdot b$
$gcd(12, 28) = 4 = -2 \cdot 12 + 1 \cdot 28$

Kan brukes for å finne multiplikativt invers
Multiplikativ inverse finnes dersom $gcd(a, b) = 1$

Finne multiplikativt invers for a med $mod\, m$

- Funker bare dersom $gcd(a, m) = 1$
- Regn ut linærkombinasjon $gcd(a, b) = s \cdot a + t \cdot b$ med gcd
- $a \cdot x \equiv 1 (mod\, m)$ er multiplicative inverse

Extended Euklids algoritme

gcd(26,7)

$(26) = 3 \cdot (7) + (5)$

$(7) = 1 \cdot (5) + (2)$

$(5) = 2 \cdot (2) + (1)$

$(2) = 2 \cdot (1) + (0)$

gcd(26,7)=1

Nå går vi tilbake:

$(5) = 2 \cdot (2) + 1$

$\implies 1 = (5) - 2 \cdot (2)$

$= (5) - 2 \cdot ((7) - (5))$

$= 3 \cdot (5) - 2 \cdot (7)$

$= 3 \cdot ((26) - 3 \cdot (7)) - 2 \cdot (7)$

$= 3 \cdot (26) - 11 \cdot (7)$

Eksempel Multiplikativt Invers

- Hva er multiplikativt invers av $7 \, mod \, 26$? ($a \cdot 7 = 1 \, mod \, 26$)
- $gcd(a, m) = gcd(7, 26) = 1 \rightarrow$ har multiplikativt invers
- Linærkombinasjon fra gcd: $3 \cdot (26) - 11 \cdot (7) = 1$
- $[1]_{26} = [3 \cdot (26) - 11 \cdot (7)]_{26} = [-11 \cdot 7]_{26}$
- $a = -11$
- $[-11]_{26} = [26 - 11]_{26} = [15]_{26} = 15$
- 15 er inverse av 7 modulo 26

Spørsmål?

Figur: Guillaume på Sandviksfjellet

Produktregelen

- Noe kan brytes ned i to aksjoner som kombineres med hverandre
- For den ene finnes det n_1 muligheter, for den andre n_2
- Det blir $n_1 \cdot n_2$ kombinasjoner

Eksempel

- Det er to type maskiner som trenges
- Den ene finnes 3 ganger, den andre 5 ganger
- Hvor mange kombinasjoner maskintype 1, maskintype 2 finnes det?
- $5 \cdot 3 = 15$

Sumregelen

- Noe kan gjøres på enten en av n_1 måter, eller en av n_2 måter
- Det finnes ingen element som er både i n_1 og i n_2
- Det blir $n_1 + n_2$ muligheter å gjøre det

Eksempel

- En student skal velge masteroppgaven sin
- Hun liker tre fagområder
- I område n_1 finnes det 5 temaer, i n_2 3 temaer, i område n_3 er det 8
- Det er $5 + 3 + 8 = 16$ temaer å velge fra

Substraksjonsregelen

- Ligner *Sumregelen*, men flere elementer er i flere grupper
- Noe kan gjøres på enten en av n_1 måter, eller en av n_2 måter, men noen er i både n_1 og n_2
- Det blir $n_1 + n_2 - felles(n_1, n_2)$ muligheter

Eksempel

- På fredag er det amerikansk-norsk folkefest
- Det er 22 amerikanere og 18 nordmenn som meldte seg på
- 3 av dem er både norsk og amerikansk
- Det er $22 + 18 - 3 = 37$ personer som deltar

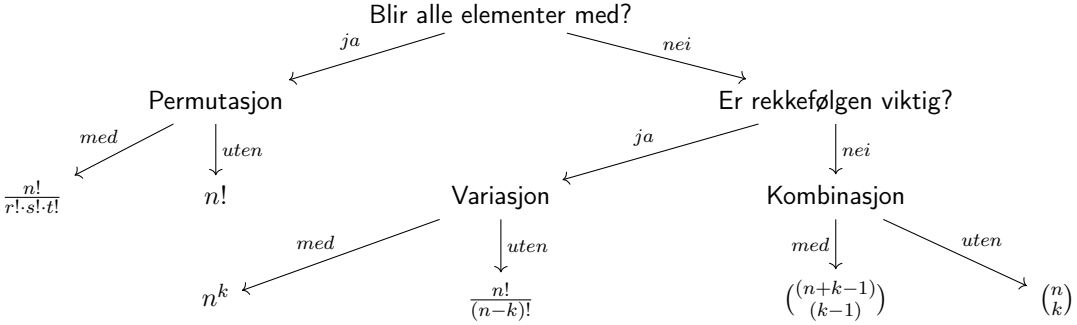
Divisjonsregelen

- Det er n måter å gjøre noe, men egentlig finnes det for hver måte minst d lignende måter
- Det blir da n/d forskjellige muligheter

Eksempel

- I en fornøyelsespark for katter blir det talt 400 bein
- Mennesker og andre dyr har ikke lov å være i fornøyelsesparken
- Hver katt har eksakt fire bein
- Det betyr det er $400/4 = 100$ katter

Permutasjon? Kombinasjon? Variasjon? Hæ?



Eksempler

Eksempel 1

- 6 personer må fotograferes. Hvor mange kombinasjoner finnes det å ordne dem på bildet?
- Alle elementer blir med, ingen repetisjon
- Permutasjon uten repetisjon: $n! = 6! = 720$

Eksempel 2

- Hvor mange måter finnes det for å ordne bokstavene *Mississippi*?
- Alle elementer blir med, men repetisjon (permutasjon)
- $\frac{n!}{r! \cdot s! \cdot t!} = \frac{11!}{4! \cdot 4! \cdot 2!} = 34650$

Enda flere eksempler

Eksempel 3

- Det er 7 personer og 3 tilfeldige av dem skal få en pris. Hvor mange kombinasjoner finnes det?
- Ikke alle elementer blir med, ingen repetisjon, rekkefølgen ikke viktig
- Kombinasjon uten repetisjon: $\binom{n}{k} = \binom{7}{3} = \frac{7!}{3! \cdot 4!} = 35$

Eksempel 4

- Vi har 5 typer is og skal spise 3 av dem. Vi er opptatt av rekkefølgen for best smak.
- Ikke alle elementer blir med, rekkefølge viktig, med repetisjon
- $n^k = 5^3 = 125$

Definisjoner

- **Utfallsrom:** Alle mengder som har en sannsynlighet
- **S:** Alle mulige utfall
- **E:** Alle ønskete utfall
- $E \subseteq S$: alle ønskete utfall er en del av alle mulige utfall
- **Sannsynlighet:** $P(E) = \frac{|E|}{|S|}$
- $0 \leq P(E) \leq 1$
- $P(S) = 1$
- $P(\overline{E}) = 1 - P(E)$

Regneregler

- $P(A \cup B) = P(A) + P(B) - P(A \cap B)$ (tilsvarer subtraction rule)
- $P(A \cap B) = P(A) \cdot P(B)$ hvis A,B statistisk uavhengig (tilsvarer multiplication rule)
- $\sum_{s \in S} P(s) = 1$ (summen av alle ting som kan skje har sannsynlighet 1)
- **Betinget sannsynlighet:** $P(A|B) = \frac{P(A \cap B)}{P(B)}$ (Sannsynlighet av A etter at B skjedde)
- **Bayes Rule:** $P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$

Eksempler

Terninger

- Det er to terninger, den ene er vanlig med 6 jevne sider
- Den andre har tallene $\{3, 4, 5, 5, 6, 6\}$
- Hvor stor er sannsynligheten at vi kaster en 7 med begge terninger?
- Det er $6 \cdot 6$ mulige kombinasjoner
- Det er fire måter å få det til:
  - Terning 1: 1, Terning 2: 6 (finnes to ganger)
  - Terning 1: 2, Terning 2: 5 (finnes to ganger)
  - Terning 1: 3, Terning 2: 4
  - Terning 1: 4, Terning 2: 3
- $P(sum = 7) = \frac{6}{36} = \frac{1}{6}$

Eksempler

Kortspill

- Vi har et vanlig tysk kortspill med 32 kort (4 farger, $\{7, 8, 9, 10, J, Q, K, A\}$)
- Hva er sannsynligheten at vi trekker et hjerte eller en konge?
- Hva er sannsynligheten at en av disse kortene er 7,8 eller 9?
- Det er 4 konger, 8 hjerter, en av dem er begge deler
- $P(Konge \cup Hjerte) = \frac{8+4-1}{32} = \frac{11}{32}$
- Blant disse er det 3 kort som er 7,8 eller 9
- $P(7, 8, 9|Konge \cup Hjerte) = \frac{P((7,8,9) \cap (Konge \cup Hjerte))}{P(Konge \cup Hjerte)} = \frac{3}{32} \cdot \frac{32}{11} = \frac{3}{11}$

Vierfeldertafel (WANTED: norsk eller engelsk begrep)

	A	$\overline{A}$	
B	$P(A \cap B)$	$P(\overline{A} \cap B)$	$P(B)$
$\overline{B}$	$P(A \cap \overline{B})$	$P(\overline{A} \cap \overline{B})$	$P(\overline{B})$
	$P(A)$	$P(\overline{A})$	1

Tabell: To hendelser A og B i en *Vierfeldertafel*

- Det er to spalter og to rekker, hver for en hendelse og motsetningen
- I alle retninger kan man summe ting sammen
- Det trenges bare tre utfylte felter for å regne ut resten

Vierfeldertafel (WANTED: norsk eller engelsk begrep)

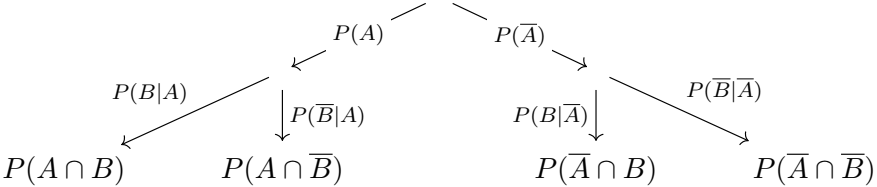
Tabell: Eksempel *Vierfeldertafel*

	A	A^C	
B	0.4		0.533
B^C		0.292	
			1

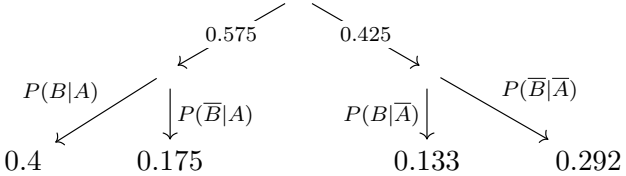
	A	A^C	
B	0.4	0.133	0.533
B^C	0.175	0.292	0.467
	0.575	0.425	1

- Det er gitt tre verdier, $P(B)$, $P(A \cap B)$, $P(A^C \cap B^C)$
- Resten kan regnes ut ved summeformalen
- Eksempel: $P(A^C \cap B) = P(A \cap B) - P(B) = 0.533 - 0.4 = 0.133$
- Kan brukes for å finne andre ting
- Betinget sannsynlighet: $P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{0.4}{0.533} = 0.75$

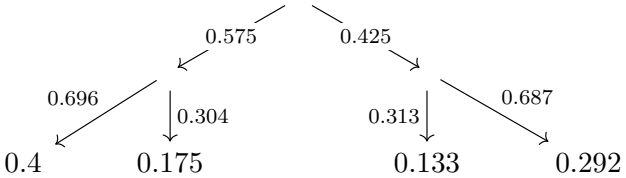
Sannsynlighetstre



Sannsynlighetstre



Sannsylighetstre



Spørsmål?

Figur: Guillaume på Fløyen

Kompleksitet

Gitt flere algoritmer som løser det samme problemet, hvordan vite hvilken som er kjappest?

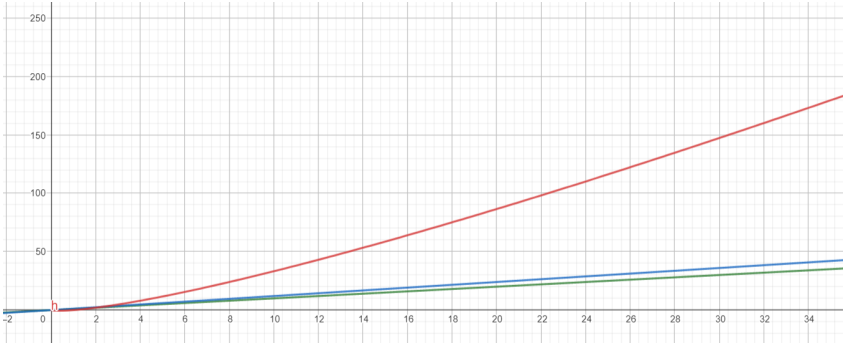
```
def minAndMax(list):
    min = list[0]
    for elem in list[1:]:
        if elem < min:
            min = elem
    max = list[0]
    for elem in list[1:]:
        if elem > max:
            max = elem
    return (min, max)
```

```
def minAndMax(list):
    min, max = list[0], list[0]
    for elem in list[1:]:
        if elem < min: min =
                            elem
        if elem > max: max =
                            elem
    return (min, max)

def minAndMax(list):
    list = sorted(list)
    return (list[0], list[-1])
```

Kompleksitet

Hvis vi hadde testet alle tre algoritmene på mange lister av økende lengde, ville det kanskje sett slik ut. Hvorfor?



Figur: Den røde er den som sorterer.

La oss si at listen har en lengde på n. Hvor mange operasjoner blir det?

```
def minAndMax(list):
    min = list[0]          # 1
    for elem in list[1:]:# n-1
        if elem < min:     # 1
            min = elem     # 1
    max = list[0]          # 1
    for elem in list[1:]:# n-1
        if elem > max:     # 1
            max = elem     # 1
    return (min, max)
```

Det blir ca

$1+(n-1)*(1+1)+1+(n-1)*(1+1)$

$= 2+4*(n-1)$

$= 4n-2,$

og det er ihvertfall mindre enn $\underline{100n}$.

Kompleksitet

```
def minAndMax(list):
    min, max = list[0], list[0]  # 2
    for elem in list[1:]:        # n-1
        if elem<min:             # 1
            min = elem           # 1
        if elem>max:             # 1
            max = elem           # 1
    return (min, max)

def minAndMax(list):
    list = sorted(list)    # n*log2(n)
    return (list[0], list[-1]) # 2
```

Den første blir $2 + (n-1) * (2+2)$
$= 2 + 4(n-1)$
$= 4n - 2$,
som er mindre enn $\underline{100n}$.

Den andre blir
$2 + n \cdot log_2(n)$,
og det er mindre enn
$\underline{100 \cdot n \cdot log_2(n)}$.

Big O

Nå skal vi formalisere konseptet om hvor godt noe skalerer.

Definition (Big O)

Gitt to funksjoner $f, g : \mathbb{N} \to \mathbb{N}$:
Hvis $\exists c, n_0 : \forall n > n_0 : [f(n) \leq c * g(n)]$, da er $f \in O(g)$.
Intuitivt kan vi tenke at $f \leq g$.

I vårt eksempel fant vi at algoritmene bruker mindre enn $100n$, $100n$, og $100 \cdot n \cdot log_2(n)$ operasjoner, og da kjører algoritmene på $O(n)$, $O(n)$, og $O(n \cdot log(n))$.

Big O er en veldig kjapp og enkel måte å estimere hva slags forskjeller som betyr noe og ikke. Konstanter spiller nesten ingen rolle i det store bildet: $O(10n) = O(5n) = O(n)$.

Regneregler for Big O

Løkker gjør at innholdet blir gjort mange ganger, da ganger vi med innholdet.

```
for _ in range(n):      # O(n)
    for _ in range(m): # O(m)
        do_something() # O(1)
# Totalt: O(n)*O(m)*O(1) = O(nm).
```

Om vi vil summere flere ting, tar vi det største. Om vi ikke vet hva som er størst beholder vi begge.

```
for _ in range(n): # O(n)
    do_something() # O(1)
do_something()     # O(1)
do_something()     # O(1)
# Totalt: O(n) + O(1) + O(1) = O(n).
```

```
for _ in range(n): # O(n)
    do_something() # O(1)
for _ in range(m): # O(m)
    do_something() # O(1)
# Totalt: O(n) + O(m) = O(n+m).
```

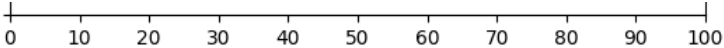
Søking

Big O er pessimistisk, og bryr seg kun om det verste tenkelige tilfellet.

```
def index_of(list, e):
    for i in range(len(list)): # O(n)
        if list[i] == e:       # O(1)
            return i           # O(1)
    return -1                  # Totalt: O(n)
```

Summen er fortsatt $O(n)$, selv om det er mulig vi blir ferdig tidligere.

Søking



Binærsøk

```
def binary_search(list, target):
    left = 0
    right = len(list) - 1
    while left <= right:
        mid = (left + right) // 2
        if list[mid] == target:
            return mid
        elif list[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

Hvor mange operasjoner blir det?
Vi må finne ut hvor mange ganger løkken looper.
I hver iterasjon blir avstanden mellom $left$ og $right$ halvert, og det kan gjøres $log_2(n)$ ganger før vi kommer til 0 eller 1.

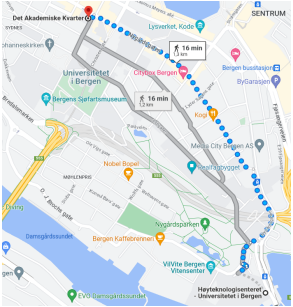
Dermed kjører et binærsøk i $log_2(n)$, som er veldig mye bedre enn $O(n)$.

Spørsmål?

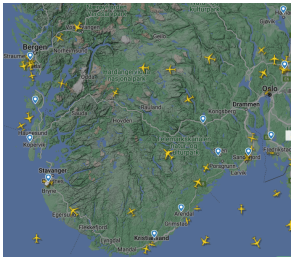
Figur: Guillaume på Ulriken

Grafer

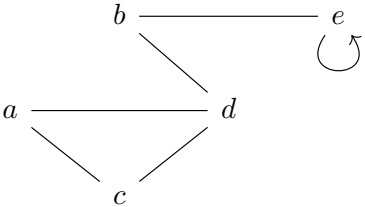
Veldig mange algoritmeproblemer kan representeres som grafer.



Figur: Hva er kjappeste veien til Kvarteret?

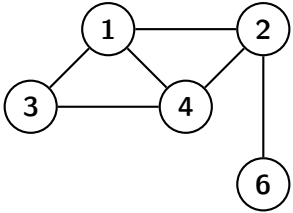


Figur: Hvilke fly kan du ta for å komme deg hjem fortest?



Graf $G(V, E)$

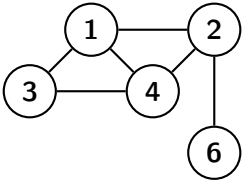
En Graf $G = (V, E)$ er et sett noder (vertices) V og et sett kanter (edges) E.



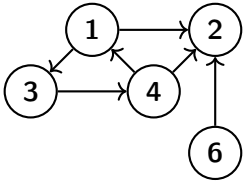
- Noder: $V = \{1, 2, 3, 4, 6\}$
- Kanter: $E = \{(1, 2), (1, 4), (3, 4), (2, 4), (2, 6), (1, 3)\}$
- Sti (/path): Vei fra A til B
  Eksempel: $Path(1, 6) = [1, 2, 6]$
- Sykel (/Cycle): En sti med samme start og slutt
  Eksempel: $[1, 3, 4, 1]$, $[1, 2, 4, 3, 1]$
- Nabolag: Alle naboene en node kan nå med én kant
  Eksempel: $N(4) = \{1, 2, 3\}$, $N(6) = \{2\}$
- Grad (/degree): Antall naboer av en node
  Eksempel: $deg(4) = 3$, $deg(6) = 1$

Rettede grafer

Om du skal lage en graf av veinettverket i Bergen, får du fort problemer. Hele sentrum er jo enveiskjørt! Vi trenger å kunne representere enveiskanter.



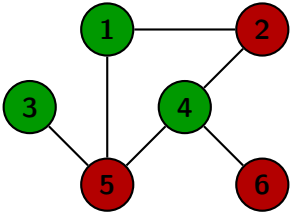
Figur: Urettet graf (undirected).
$deg(4) = 3, N(4) = [1, 2, 3]$



Figur: Rettet graf (directed).
$deg^+(4) = deg^{out}(4) = 2$
$deg^-(4) = deg^{in}(4) = 1$
$N^+(4) = N^{out}(4) = [1, 2]$
$N^-(4) = N^{in}(4) = [3]$

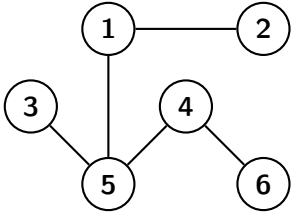
Bipartitte grafer $G(V, A, B)$

En bipartitt graf er en graf som kan deles i to sett A, B der alle kanter $v \in V$ går fra en node i A til en node i B. Grafen kan fargelegges i to farger slik at alle kanter går fra én farge til en annen farge.

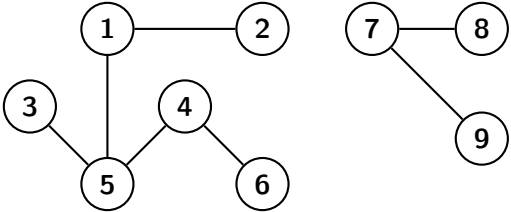


Trær

- Et *tre* er en *sammenhengende*, *urettet* graf uten sykler.
- En *skog* er en graf med flere trær som ikke er tilknyttet hverandre.
- Et *blad* i et tre er en node som har grad 1.
- En *intern node* i et tre er en node som ikke er et blad.



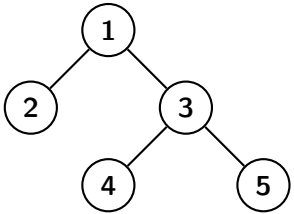
Figur: Et tre



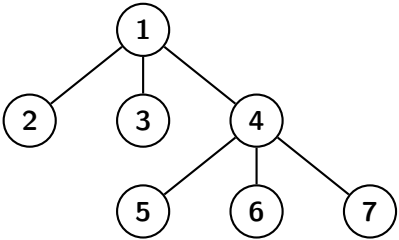
Figur: En skog

Rotfestede trær

- Et *rotfestet* tre er et tre med en dedikert rotnode.
- Nodene 'under' en annen node kalles for *barnene* til noden.
- Noden 'over' en node kalles for forelderen. Alle unntatt roten har en forelder.
- Et binærtre er et rotfestet tre der alle har enten 0, 1 eller 2 barn.



Figur: binary tre



Figur: 3-ary tre

Måter å oppgi relasjoner på

$$\begin{tikzcd} A \arrow[r] \arrow[d] \arrow[dr] & B \arrow[d] \arrow[dl] \\ D & C \arrow[l] \end{tikzcd}$$

Kantliste:
$\{(A,B),(A,C),(A,D),(B,C),(B,D),(C,D)\}$

Settkomprehensjon:
$\{(x,y)|x,y \in \{A,B,C,D\}, x < y\}$:

Matrise:

	A	B	C	D
A	0	1	1	1
B	0	0	1	1
C	0	0	0	1
D	0	0	0	0

Nabolister:
$N(A) = [B,C,D]$
$N(B) = [C,D]$
$N(C) = [D]$
$N(D) = []$

Måter å oppgi ~~relasjoner~~ grafer på

$$\begin{tikzcd} A \arrow[r] \arrow[d] \arrow[dr] & B \arrow[d] \arrow[dl] \\ D & C \arrow[l] \end{tikzcd}$$

Kantliste:
$\{(A,B),(A,C),(A,D),(B,C),(B,D),(C,D)\}$

Settkomprehensjon:
$\{(x,y)|x,y \in \{A,B,C,D\}, x < y\}$:

Matrise:

	A	B	C	D
A	0	1	1	1
B	0	0	1	1
C	0	0	0	1
D	0	0	0	0

Nabolister:
$N(A) = [B,C,D]$
$N(B) = [C,D]$
$N(C) = [D]$
$N(D) = []$

Søk

Om vi har en graf, er det hovedsakelig to måter vi kan søke igjennom grafen:

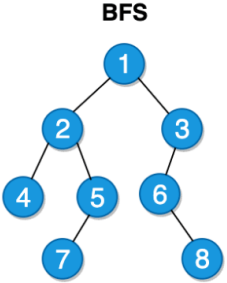
- Bredde-Først Søk (BFS)
- Dybde-Først Søk (DFS)

Forskjellen ligger i når hver node blir søkt.

BFS: Bredde-Først-Søk

Vi holder styr på hvilken 'generasjon' med noder vi jobber med. For hver generasjon ser vi etter alle noder som den generasjonen kan nå, og det er neste generasjon.

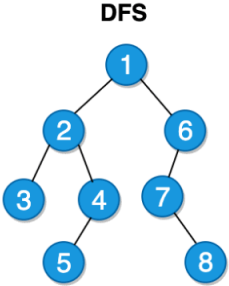
```
def bfs(start, graph):
    visited = { u: False for u in graph }
    visited[start] = True
    current_gen = [start]
    next_gen = []
    while len(current_gen) > 0:
        for u in current_gen:
            print(u)
            for v in graph[u]:
                if not visited[v]:
                    next_gen.append(v)
                    visited[v] = True
        current_gen, next_gen = next_gen, []
```



DFS: Dybde-Først-Søk

Her går vi heller i dybden først. Hver gang vi oppdager en ny node søker vi den umiddelbart før vi går tilbake igjen.

```
def dfs(start, graph, visited):
    print(start)
    visited[start] = True
    for v in graph[start]:
        if not visited[v]:
            dfs(v, graph, visited)
```



Korteste sti

Vi kan bruke BFS til å finne den korteste stien i en graf, og dermed også den korteste ruten til Kvarteret. Men noen kanter/veier tar lengre tid å gå enn andre, så i praksis fungerer det ikke så bra.

$$\begin{tikzcd}
\text{Høytek} \arrow[r] \arrow[d] & B \arrow[r] \arrow[d] & \text{Kvarteret} \\
A \arrow[r] & C \arrow[r] & D \arrow[u]
\end{tikzcd}
\qquad
\begin{tikzcd}
\text{Høytek} \arrow[r, "2"] \arrow[d, "1"] & B \arrow[r, "8"] \arrow[d, "3"] & \text{Kvarteret} \\
A \arrow[r, "5"] & C \arrow[r, "1"] & D \arrow[u, "2"]
\end{tikzcd}$$

Definition (Vektet graf)

En vektet graf er en der alle kantene også har en vekt/kostnad.

Dijkstras algoritme

Dijkstras algoritme fungerer utmerket for å finne den korteste stien i en vektet graf, så lenge vektene er positive tall.

```
def dijkstra(start, end, graph):
    dist = { u: 99999999999999 for u  in graph }
    dist[start] = 0
    queue = [start]
    while len(queue) > 0:
        next = min(queue, key=lambda u: dist[u]) # finner noden med lavest distanse
        queue.remove(next)
        for (v, weight) in graph[next]:
            if dist[v] < dist[next] + weight:
                dist[v] = dist[next] + weight
                queue.append(v)
    return dist
```

Spørsmål?

Figur: Guillaume foran Tvindefossen

Lykke til på eksamen!

Takk for oss :)