MAT1030 – Diskret Matematikk

Forelesning 25: Trær

Dag Normann

Matematisk Institutt, Universitetet i Oslo

27. april 2010

(Sist oppdatert: 2010-04-27 14:15)

Forelesning 25

Litt repetisjon

- Vi har snakket om grafer og trær. Av begreper vi så på var følgende:
- Eulerstier og Eulerkretser
- Hamiltonstier og Hamiltonkretser
- Trær
- Vektede grafer
- Minimale utspennende trær

Litt repetisjon

- Et tre er en sammenhengende graf uten sykler.
- En sykel er en krets hvor samme kant ikke kan forekomme to ganger og hvor samme node ikke kan forekomme to ganger.
- Dette siste kravet ble uteglemt forrige uke, uten at det endrer hva vi mener med et tre.
- Et tre kan faktisk defineres som en sammenhengende graf hvor det finnes én node mer enn antall kanter.
- Vi viste at dette er en egenskap ved trær.
- Omvendingen gis som en utfordring til de som har lyst til å prøve seg på et bevis.

Litt repetisjon

- Vi minner om at en *vektet graf* er en enkel graf hvor hver kant har en vekt, et positivt reelt tall (Alternativt: Et ikke-negativt 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 vet at hvis G har n noder, vil et utspennende tre ha $n - 1$ kanter.
- Omvendt, vil et tre med $n - 1$ kanter ha n noder.
- 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.
- Forrige uke så vi på det som kalles Prims algoritme.
  Vi tar den fra begynnelsen igjen.

Prims 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 først 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 annerledes enn den er formulert i læreboka, men effekten er den samme, vi får det samme treet bygget opp i den samme rekkefølgen.

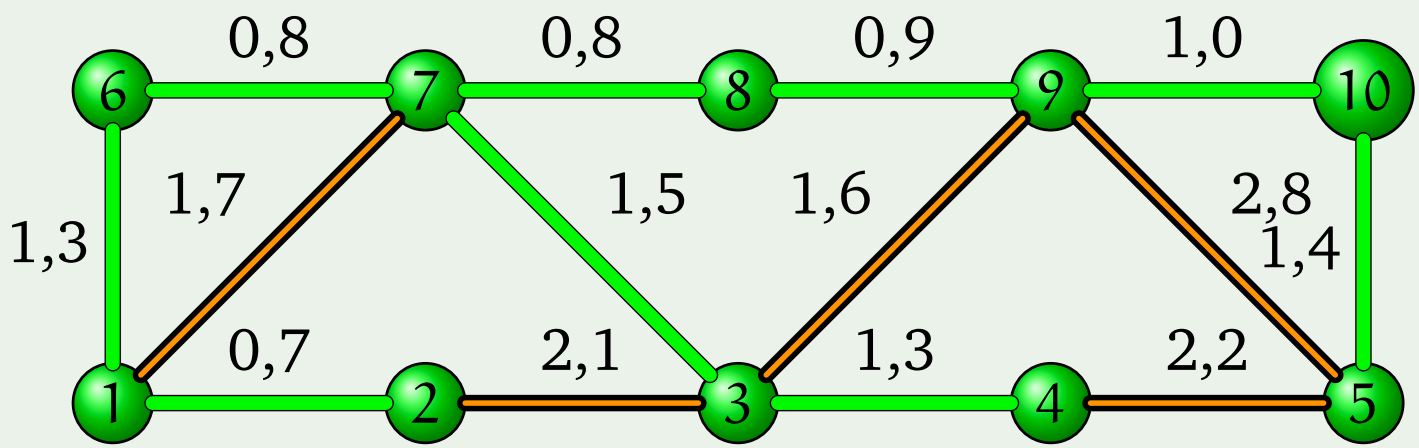
Prims algoritme

- La G være en vektet, sammenhengende graf med noder $V = \{v_1, \ldots, v_n\}$.
- La T_1 være treet som består av v_1 og ingen kanter.
- Start med node v_1 og la $V_1 = \{v_2, \ldots, v_n\}$, 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.
- 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.

Prims algoritme

Eksempel (Fortsatt)

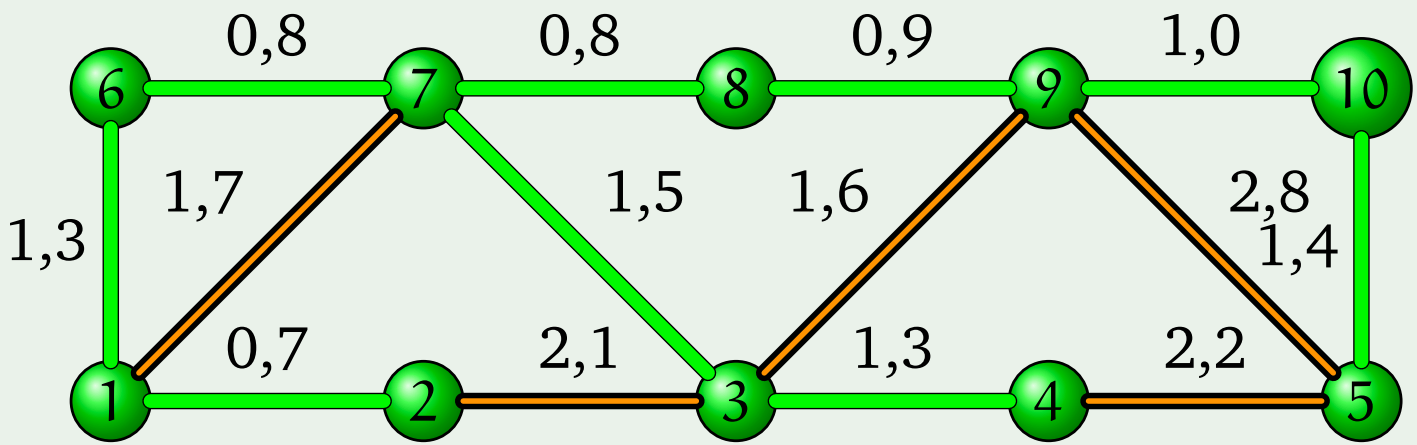
- Vi ser på hvordan man ved hjelp av Prims algoritme, skritt for skritt, kan bygge opp et utspennende tre med minimal vekting.
- Vi starter i Node 1.



Prims algoritme

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.

Prims algoritme

- Vi gir en pseudokode for Prims algoritme.
- Den ser litt annerledes ut enn den som står i boka, men effekten, skritt for skritt, er den samme.

1 *Input* $V = \{v_1, \ldots, v_n\}$ [Nodene i G]

2 *Input* $E = \{e_1, \ldots, e_k\}$ [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 $E \leftarrow E - \{e_i \ ; \ e_i$ *ligger inntil to noder i* $T\}$.

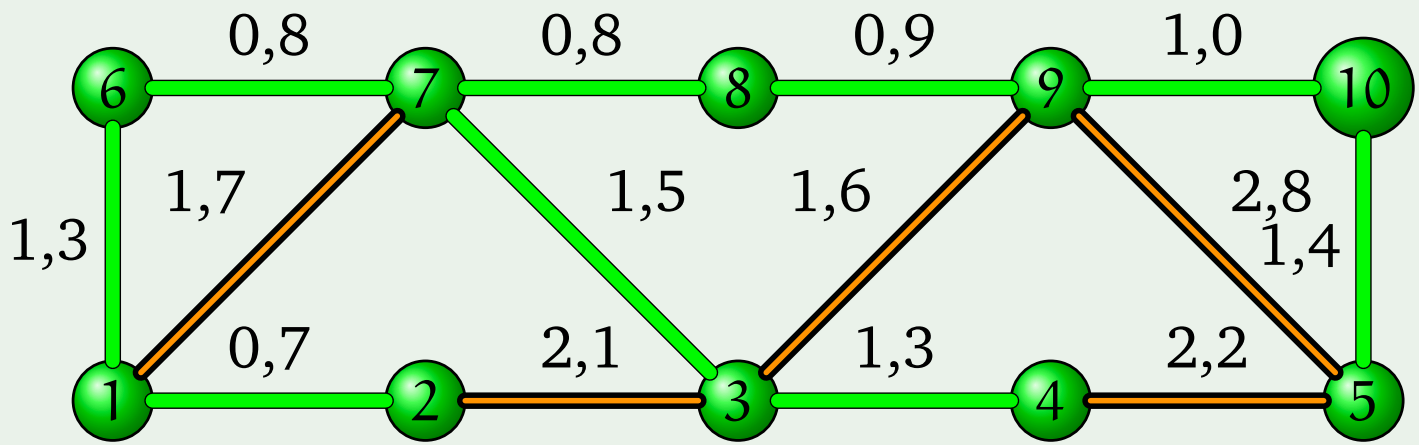
6 *Output* (T, K).

Prims algoritme

- Det fins 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.
- Fins 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.

Prims algoritme

Eksempel (Fortsatt)



- Vi får fortsatt det samme treet.
- I utgave 3 blir denne algoritmen omtalt (i oppgavedelen) som Kruskals algoritme.

Prims algoritme

Vi regner et annet eksempel på tavla, både ved bruk av Prims algoritme og Kruskals algoritme.

Dijkstras algoritme

- Et annet naturlig problem i forbindelse med vektede 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 fins 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".

Dijkstras algoritme

- 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 $i \leqslant n$ 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 v_1 til v, via T_i og e, på denne måten.
  2. Fins 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.

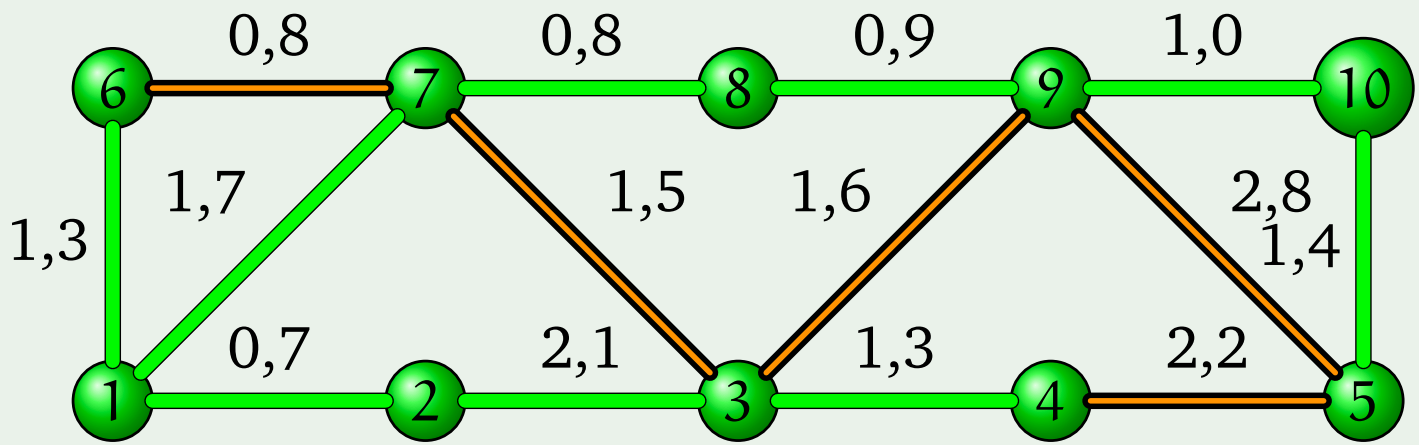
Dijkstras algoritme

- 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.

Dijkstras algoritme

Eksempel

- Vi starter i Node 1
- Vi får et nytt tre.



Dijkstras algoritme

- Vi illustrerer Dijkstras algoritme ved to eksempler på tavla.
- Noen har lært andre måter å gjennomføre Prims algoritme, Kruskals algoritme og Dijkstras algoritme på enn slik vi gjør det her.
- Bruker dere en alternativ metode, må dere forklare den for sensor.

Dijkstras algoritme

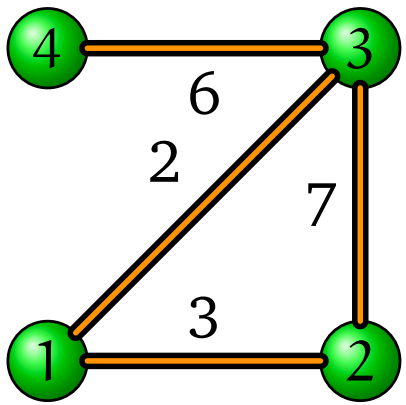
- Dijkstras algoritme kan også brukes til å finne korteste sti mellom to noder.
- Det vil være stien mellom noedne med minst mulig samlet vekt.
- Vi lar en av nodene være startnoden i Dijkstras algoritme, og så følger vi algoritmen til den andre noden er med i treet vi konstruerer.

Representasjoner

- Når vi beskriver algoritmer som finner bestemte typer trær i vektede 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 vektede 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.
- Vektede grafer kan også representeres som matriser.
- Vi følger boka, og lar ∞ representere at det ikke fins 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 et eksempel.

Representasjoner

En vektet graf.



Matriserepresentasjonen.

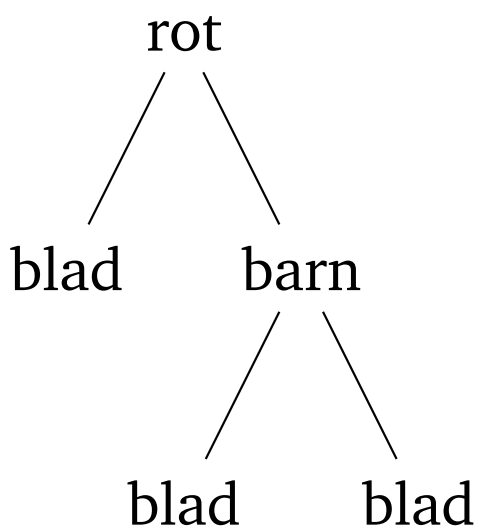
	1	2	3	4
1	0	3	2	∞
2	3	0	7	∞
3	2	7	0	6
4	∞	∞	6	0

Denne representasjonen må ikke forveksles med matriserepresentasjonen av grafer som ikke er enkle.

Trær med rot

- Til nå har vi sett på trær som sammenhengende grafer uten sykler.
- 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 annerledes enn trær uten rot.

Trær med rot

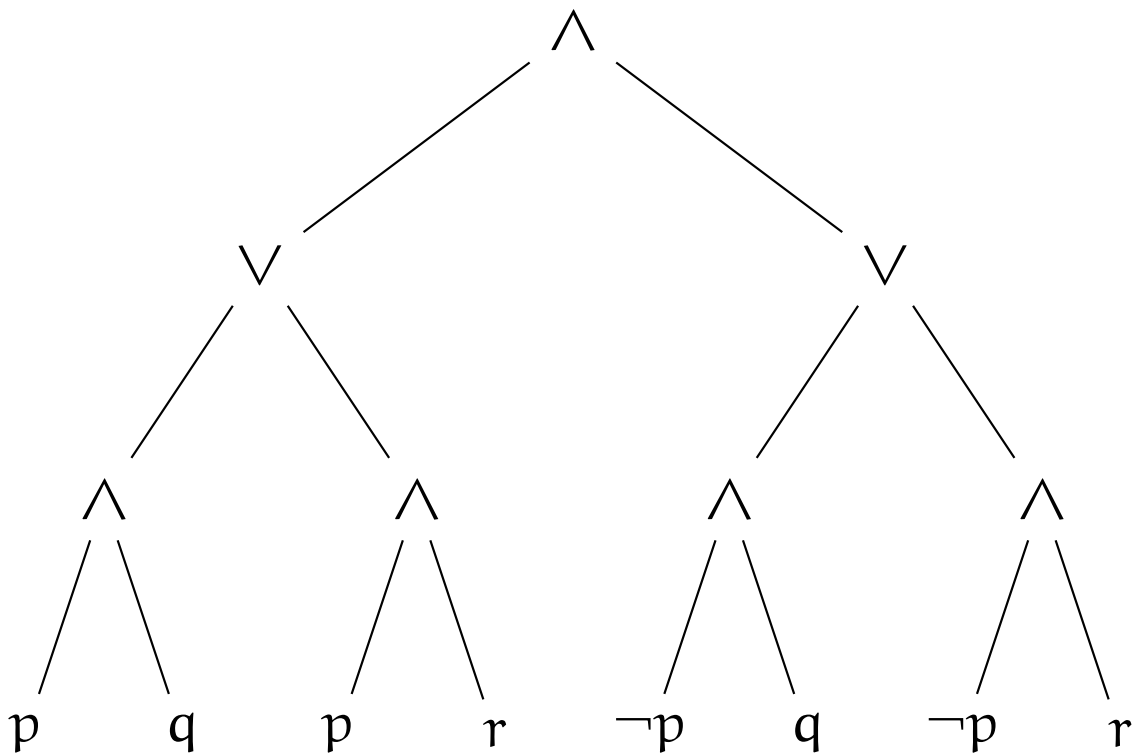


- Roten tegnes øverst, og treet "vokser" nedover.
- Nodene ligger i "lag" avhengig av avstanden til roten.
- Vi kan snakke om barna til en node, og om bladene til et tre med rot.

Vi skal se på endel eksempler før vi går nærmere inn på terminologien. Det er ikke noe i veien for å tegne roten nederst eller til venstre eller høyre på arket, det er bare ikke vanlig.

Trær med rot

- 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:

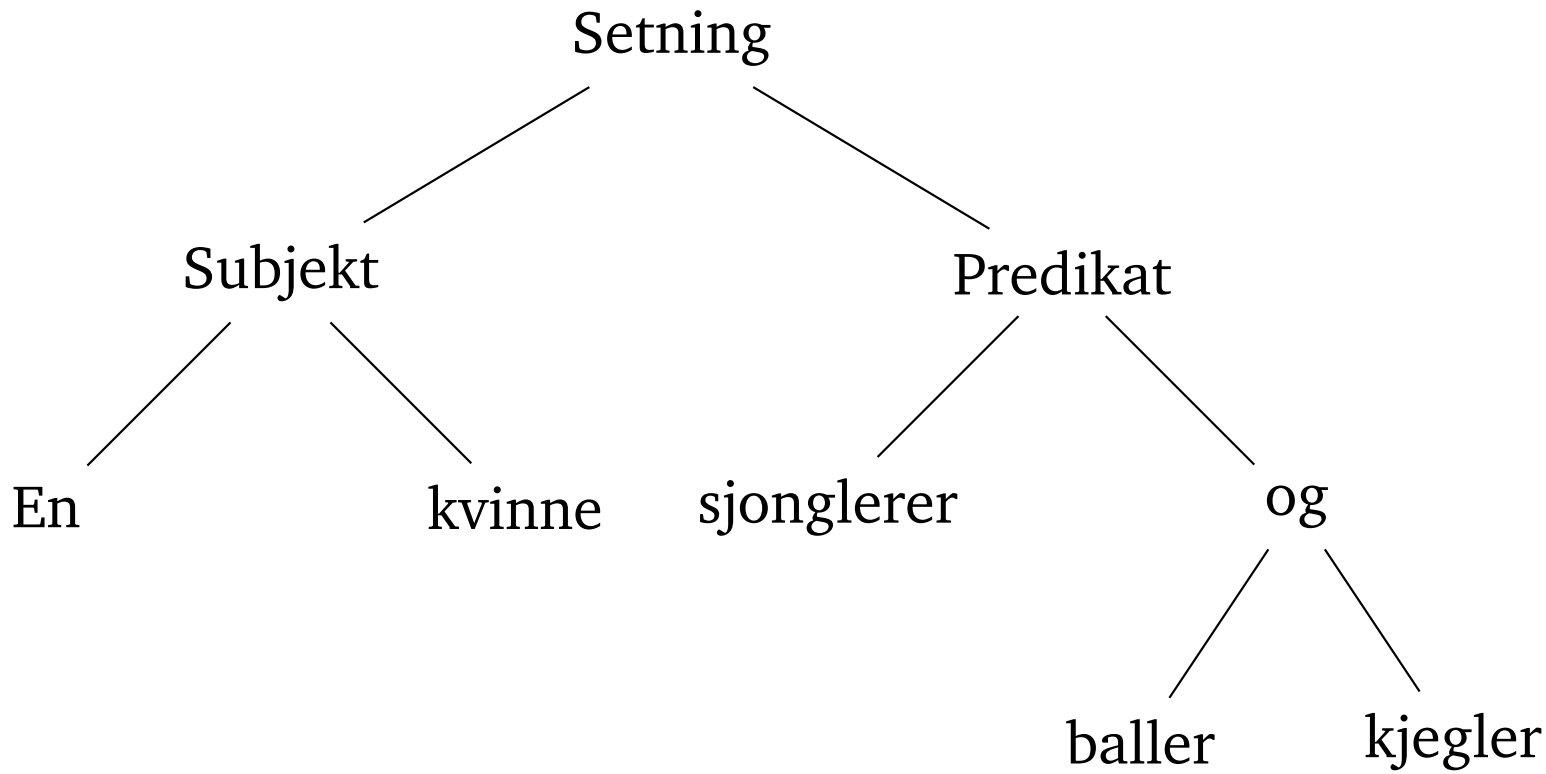


Trær med rot

- 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 grammatikkanalyse av setninger i naturlige språk.
- Da setter man opp et tre som beskriver hvordan en setning f.eks. 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.

Trær med rot

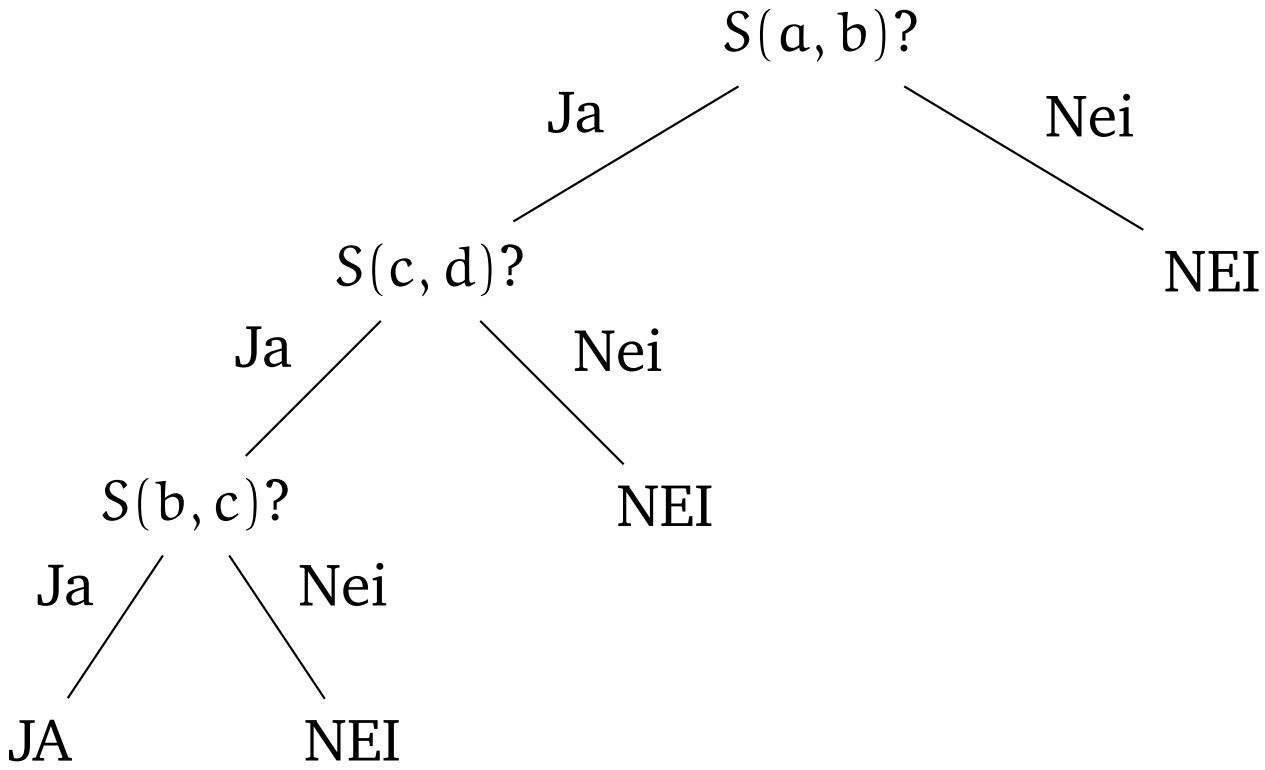
- *En kvinne sjonglerer baller og kjegler.*
- Denne setningen er bygget opp slik.



Trær med rot

- 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 og 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 utelukkende på dét (på samme måte som korrekthet av eksemplet i boka bygger på at vi har en total ordning).

Trær med rot



Trær med rot

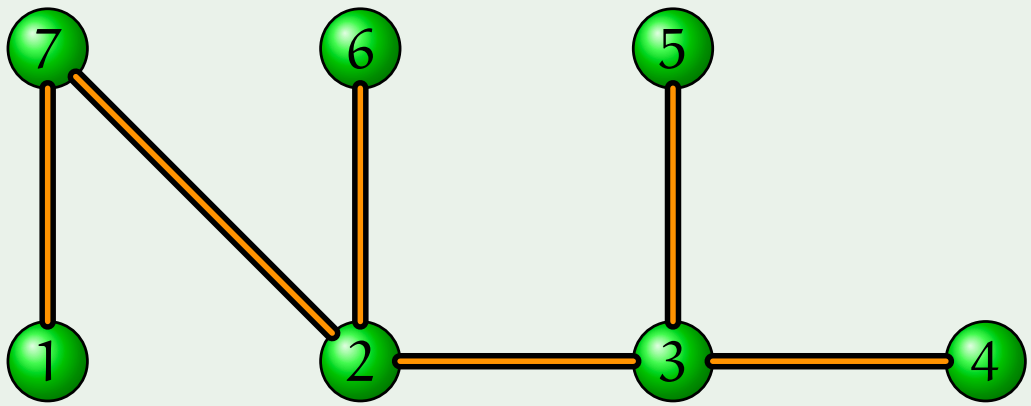
- Det er verd å merke seg at prosedyren over faktisk sjekker om a, b, c og d står i relasjon S til hverandre når S er en transitiv relasjon.
- Vi sjekker først $S(a, b)$.
- Får vi negativt svar gir algoritmen negativt svar.
- Får vi positivt svar sjekker vi $S(c, d)$.
- Får vi positivt svar her også sjekker vi til slutt $S(b, c)$.
- Hvis S er transitiv vet vi at a, b, c og d ligger etter hverandre i S-relasjonen.
- 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.

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:



Trær med rot

Definisjon

- La T være et tre med rot (anta at vi tegner T med roten øverst).
- Med nivået til en node mener vi antall kanter mellom noden og roten.
- Hvis det fins en kant mellom node a og b, og a har lavest nivå (ligger øverst i tegningen) sier vi at b er barnet til a.
- Hvis det fins en sti mellom to noder slik at
  - en kant k i stien ligger inntil nodene a og b (det vil si at sekvensen akb er en del av stien) nøyaktig når b er barnet til a,

  så vil den som har det høyeste nivået (ligger nederst i tegningen) være etterkommer til den andre, som omvendt er forgjenger til den første.

Trær med rot

Definisjon (Fortsatt)

- En node som ikke har noen barn er et blad eller en løvnode.
- 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.