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.

Oppsummering

- 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.

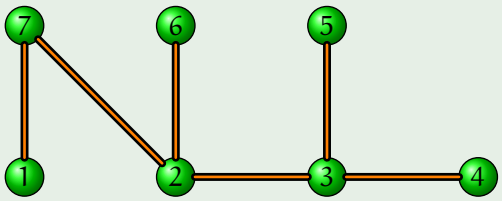
Oppsummering

- 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

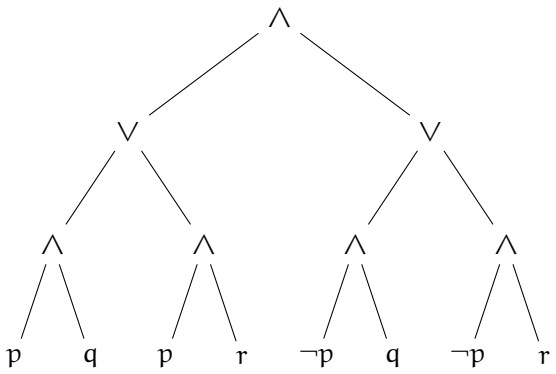
Hvis vi tar utgangspunkt i et tre uten rot, og så tegner det som et tre med rot, vil det visuelle resultatet avhenge mye av hvor vi plasserer roten.

Oppgave. • Tegn følgende tre som et tre med rot når vi plasserer roten henholdsvis i nodene 1, 3 og 6:



Trær med rot

- Mandag ga vi et eksempel på et syntakstre hvor vi løste opp et utsagnslogisk uttrykk i en trestruktur.



Trær med rot

- Når vi skriver utsagnslogiske uttrykk eller algebraiske uttrykk, er vi vante 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.

Trær med rot

- 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 lommeregnere programmeres med baklengs polsk notasjon.
- Brukere av editoren *emacs* vil oppdage at den innebygde kalkulatoren bruker baklengs polsk notasjon.

Trær med rot

- 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.

Trær med rot

- 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

  $t ::= 0 \mid 1 \mid +tt \mid \times tt$

Trær med rot

- Denne definisjonen skal leses som følger:
- Mengden av *termer* er den minste mengden som oppfyller
  - 0 og 1 er termer.

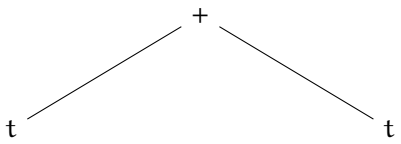
- Hvis t og s er termer, er +ts og ×ts også termer.

- 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?

Trær med rot

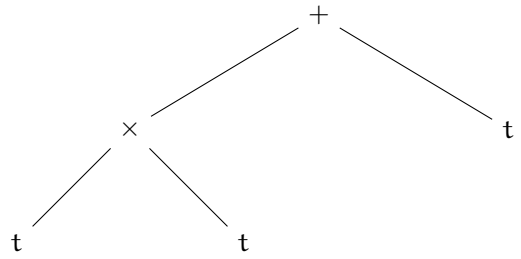
- Er $+ \times 00 \times +100$ en term slik det ble definert på forrige side?
- Vi kan prøve å utvikle et syntakstre for dette ordet, hvor vi bruker bokstaven t for å markere at her må det stå en enklere term.
- Første tilnærming til syntakstreet må være



  hvor $tt = \times 00 \times +100$

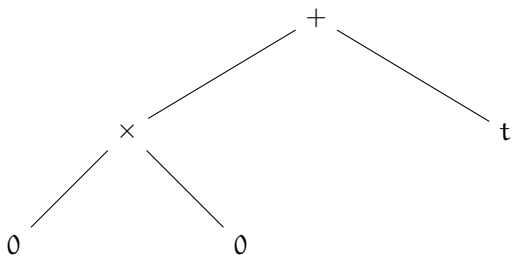
Trær med rot

- 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

Trær med rot



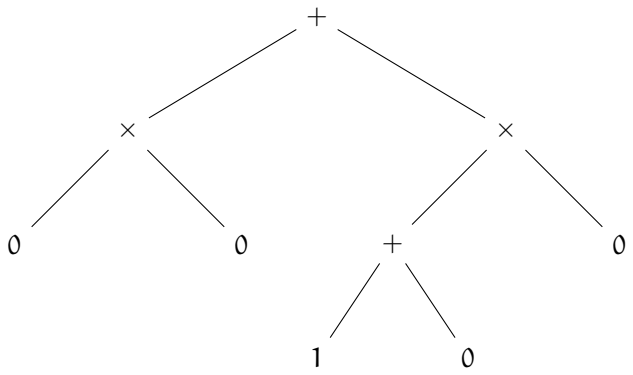
hvor $t = \times + 100$.

Trær med rot

- 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:

Trær med rot



Skrevet på vanlig infiks-form får vi

$$(0 \times 0) + ((1 + 0) \times 0).$$

Trær med rot

- 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.

Trær med rot

- *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.

Trær med rot

- 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.

Trær med rot

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.

Trær med rot

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.

Trær med rot

- 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.

Trær med rot

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:

Binære trær

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

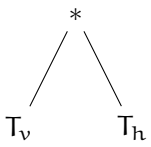
Binære trær

- 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.

Binære trær

- 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.

Binære trær



Sammensetning av to binære trær til ett.

Noen konkrete eksempler vises på tavla.

Binære trær

- 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 kommuttative 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.

Binære trær

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.

Binære trær

- Det er en intim sammenheng mellom bitsekvenser og binære trær.
- For hver node i et binært tre kan vi lage oss en tilsvarende bitsekvens ved rekursjon på nivået (avstanden til roten) til noden:
- $\mathtt{bit}(*)$ er den tomme sekvensen hvis $*$ er rotnoden.
- Hvis a er en node med to barn, b til venstre og c til høyre, og $\mathtt{bit}(a) = x_1 \cdots x_k$, lar vi
  - $\mathtt{bit}(b) = x_1 \cdots x_k 0$.
  - $\mathtt{bit}(c) = x_1 \cdots x_k 1$.

  Denne definisjonen illustreres på tavla.

Binære trær

- 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.