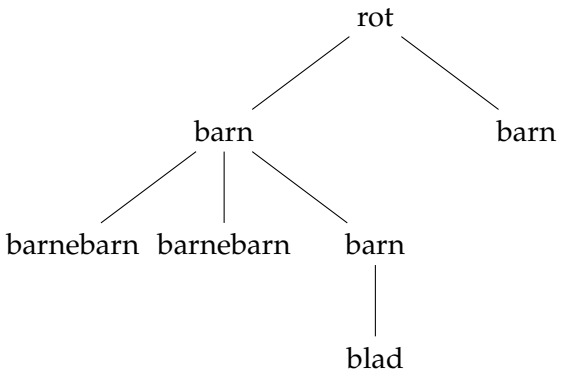
Forelesning 26

Trær

Dag Normann - 28. april 2008

Oppsummering

- Sist forelesning snakket vi i hovedsak om trær med rot, og om praktisk bruk av slike.

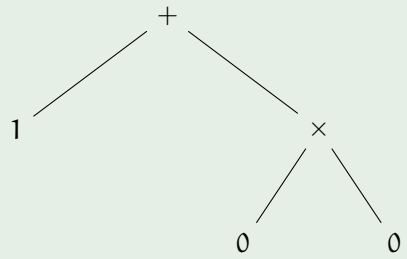


- Her er noen av de begrepene vi innførte.

- 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 $t + s$.

Eksempel.

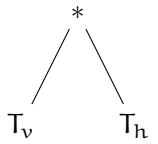


- Vanlig: $1 + (0 \times 0)$
- Polsk: $+1 \times 00$
- Baklengs polsk $100 \times +$

- 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 $T =$

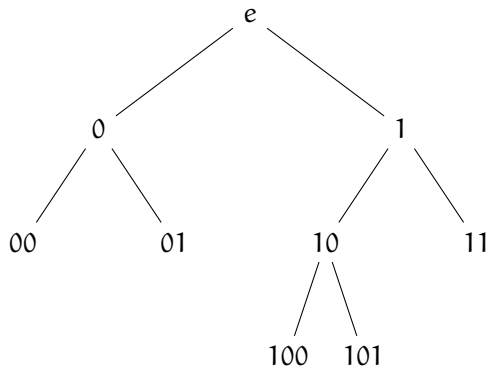


- Vi kan da definere en funksjon f ved rekursjon over oppbyggingen av binære trær ved
  - Bestemme hva $f(*)$ er.
  - Bestemme hvordan $f(T)$ avhenger av de to deltrærne $f(T_v)$ og $f(T_h)$når T er et sammensatt tre.

- 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 strippet 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.

a) 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.

b) 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 $\mathtt{infiks}$ som gir oss den vanlige måten å skrive en term på,
  - funksjonen $\mathtt{polsk}$ som gir oss termen med polsk notasjon,
  - funksjonen $\mathtt{revpolsk}$ som gir oss termen på baklengs polsk form,
- og vi skal vise ved eksempler på tavlen hvordan disse rekursjonene virker.

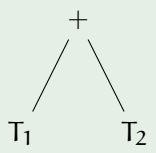
Eksempel.

Vi definerer funksjonen $\mathtt{infiks}(T)$ ved trerekursjon ved

- Hvis roten i T er en bladnode med merke a ($a = 0$, $a = 1$ eller $a = -1$) lar vi

$$\mathtt{infiks}(T) = a$$

- Hvis T er på formen



lar vi

$$\mathtt{infiks}(T) = (\mathtt{infiks}(T_1) + \mathtt{infiks}(T_2)).$$

- Tilsvarende for $\times$.

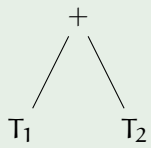
Eksempel.

Vi definerer funksjonen $\mathtt{polsk}(T)$ ved trerekursjon ved

- Hvis roten i T er en bladnode med merke a ($a = 0$, $a = 1$ eller $a = -1$) lar vi

$$\mathtt{polsk}(T) = a$$

- Hvis T er på formen



lar vi

$$\mathtt{polsk}(T) = +\mathtt{polsk}(T_1)\mathtt{polsk}(T_2).$$

- Tilsvarende for $\times$

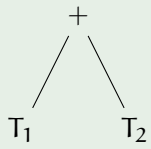
Eksempel.

Vi definerer funksjonen $\mathtt{revpolsk}(T)$ ved trerekursjon ved

- Hvis roten i T er en bladnode med merke a ($a = 0$, $a = 1$ eller $a = -1$) lar vi

$$\mathtt{revpolsk}(T) = a$$

- Hvis T er på formen



lar vi

$$\mathtt{revpolsk}(T) = \mathtt{revpolsk}(T_1)\mathtt{revpolsk}(T_2) + .$$

- 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 $\texttt{polsk}(T)$ 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 $\texttt{polsk}(T_v)$ og $\texttt{polsk}(T_h)$.
- 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.

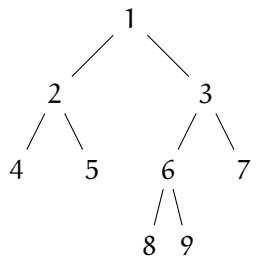
- Funksjonen $\texttt{revpolsk}$ er basert på at vi leser nodene i syntakstreet i den samme rekkefølgen, ovenfra og ned, og barnenodene før søskennodene, men ellers, fra venstre mot høyre.
- Siden merket på roten skal skrives til sist, må dette merket lagres, normalt i noe vi kaller en stack, og dette symbolet skriver vi bare ut når vi er helt ferdige med resten.
- Da må vi selvfølgelig lagre informasjon under beregningen av $\texttt{revpolsk}(T_v)$ og av $\texttt{revpolsk}(T_h)$ 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 $\texttt{infiks}$ 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 $\texttt{infiks}(T)$ 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 parallellprosessor, 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.