MAT1030 – Diskret Matematikk

Forelesning 26: Trær

Dag Normann

Matematisk Institutt, Universitetet i Oslo

28. april 2010

(Sist oppdatert: 2010-04-28 12:57)

Forelesning 26

Litt repetisjon

- Prims algoritme
  - finne det minste utspennende treet i en vektet graf
  - en *grådig* algoritme i den forstand at den vurderer lokalt hva som er det beste neste skrittet
- Dijkstras algoritme
  - en av nodene er *sentrum*
  - finne det treet som gir kortest mulig vei fra hver av de andre nodene til sentrum
- Matriserepresentasjoner
- Trær med rot
  - en av nodene har status som *rot*

Trær med rot

- La T være et tre med rot (anta at vi tegner T med roten øverst).
- Vi definerte:
- Nivået til en node.
- Hva vi mener med et barn til en node.
- Hva vi mener med en etterkommer til en node.
- Hva vi meder med en forgjenger til en node.
- En node som ikke har noen barn er et blad eller en løvnode.
- En gren er en sti fra roten til et blad.

Binære trær

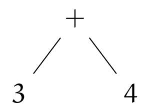
- Veldig mange trær har den egenskapen at hvis en node ikke er en bladnode, så har den nøyaktig to barn.
- Vi skiller ut disse ved en egen betegnelse.

Definisjon

- Et binært tre er et tre med rot slik at følgende holder.
  1. Enhver node er enten en bladnode eller har nøyaktig to barn.
  2. Hvis en node har to barn, vil det ene barnet betegnes som barnet til venstre og det andre som barnet til høyre.
- Det venstre deltreet får vi ved å fjerne roten og barnet til høyre og alle dets etterkommerne. Tilsvarende for det høyre deltreet. (I et deltre blir det ene barnet den nye roten.)
- I et tre med kun én node kan vi ikke snakke om deltrær.

Traverseringer

- En traversering av et tre innebærer at vi leser nodene i treet i en bestemt rekkefølge og utfører operasjoner (som å skrive symboler) i en bestemt rekkefølge.
- Eksempel



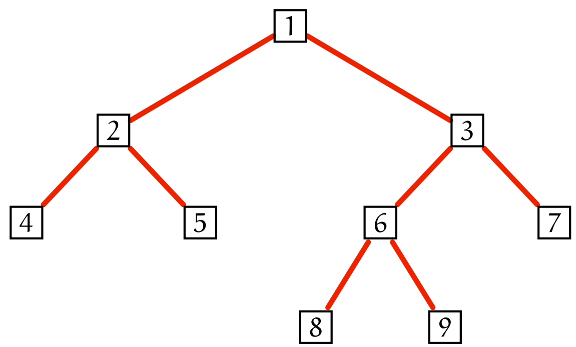
- Vi skal se på tre vanlige måter å traversere et tre på.
  - *in-order* traversering – svarer til *infiks* notasjon: $3 + 4$
  - *pre-order* traversering – svarer til *prefiks* notasjon: $+34$
  - *post-order* traversering – svarer til *postfiks* notasjon: $34+$

In-order-traversering

- Her er algoritmen for den traverseringen som gir *infiks* notasjon hvis input er et syntakstre.

Algoritme *in-order_traverse*(T):

1. **If** T ikke er et blad **then**
   1.1. *in-order_traverse*(*venstre deltre av* T)
2. Output roten til T
3. **If** T ikke er et blad **then**
   3.1. *in-order_traverse*(*høyre deltre av* T)



in-order-traversering gir 4 2 5 1 8 6 9 3 7

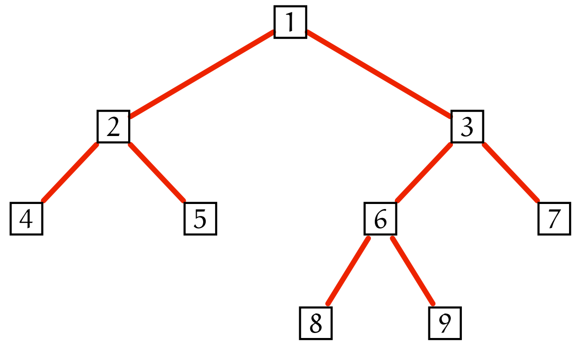
Pre-order-traversering

- Her er algoritmen for den traverseringen som gir *polsk prefiks* notasjon hvis input er et syntakstre.

Algoritme *pre-order_traverse*(T):

1. Output roten til T
2. **If** T ikke er et blad **then**
   2.1. *pre-order_traverse(venstre deltre av* T*)*
3. **If** T ikke er et blad **then**
   3.1. *pre-order_traverse(høyre deltre av* T*)*

Pre-order-traversering



pre-order-traversering gir 1 2 4 5 3 6 8 9 7

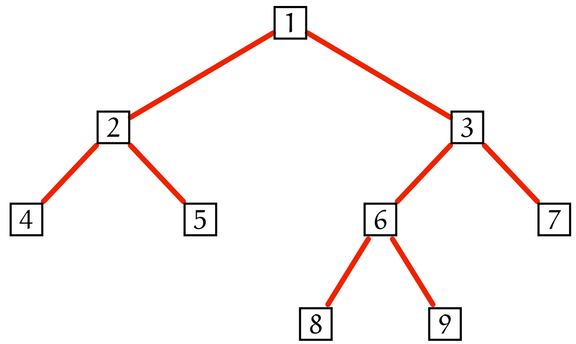
Post-order-traversering

- Her er algoritmen for den traverseringen som gir *baklengs polsk* notasjon, eller *postfiks* notasjon, hvis input er et syntakstre.

Algoritme *post-order_traverse*(T):

1. **If** T ikke er et blad **then**
   1.1. *post-order_traverse(venstre deltre av* T*)*
2. **If** T ikke er et blad **then**
   2.1. *post-order_traverse(høyre deltre av* T*)*
3. Output roten til T

Post-order-traversering



post-order-traversering gir 4 5 2 8 9 6 7 3 1

Notasjon: infiks, prefiks, postfiks

- Når vi skriver utsagnslogiske eller algebraiske uttrykk, er vi vant til å skrive symbolene $\wedge, \vee, +, \cdot$, etc. *mellom* deluttrykkene.
- Denne skrivemåten er historisk betinget og kalles for infiks.
- Fordelen med forlengs og baklengs polsk notasjon, eller *prefiks* og *postfiks*, er at man slipper parenteser.
- Disse kan være bedre egnet for innmating i algoritmer.
- 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 (RPN).

Eksempel

- Vi skal gi en grammatikk som definerer alle termer i symbolene 0, 1, $+$ og $\times$ i forlengs polsk, eller prefiks, notasjon.
- En term er et uttrykk som beskriver et tall. Vi skal se på alle måter å beskrive tall på hvor vi bruker symbolene nevnt over.
- Poenget med polsk notasjon er at funksjonssymbol, logiske bindeord og andre symboler vi bruker til å binde sammen enkle uttrykk til mer komplekse settes først, og så kommer deluttrykkene etter, uten parenteser.
- En grammatikk er et sett regler som forteller oss hvilke ord som tilhører det formelle språket vi vil beskrive.
- I informatikk-litteratur har man, som tidligere nevnt, utviklet en rask skrivemåte for slike grammatikker.

  Term t

  $\mathrm{t} ::= 0 \mid 1 \mid +\mathrm{tt} \mid \times\mathrm{tt}$

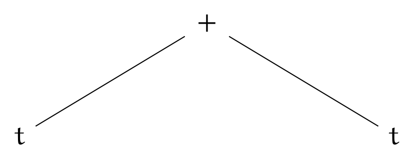
Eksempel

- 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 $+\mathrm{ts}$ og $\times\mathrm{ts}$ 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?

Eksempel

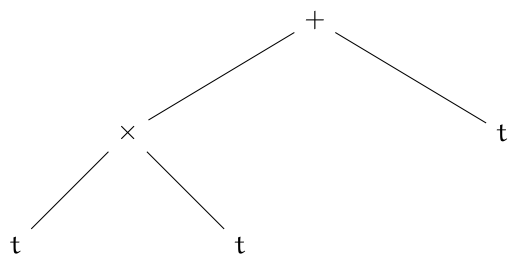
- Er $+\times 00 \times +100$ en term slik det ble definert på forrige side?
- Vi kan prøve å utvikle et syntakstre for dette ordet, hvor vi bruker bokstaven t for å markere at her må det stå en enklere term.
- Første tilnærming til syntakstreet må være



hvor $\mathrm{tt} = \times 00 \times +100$

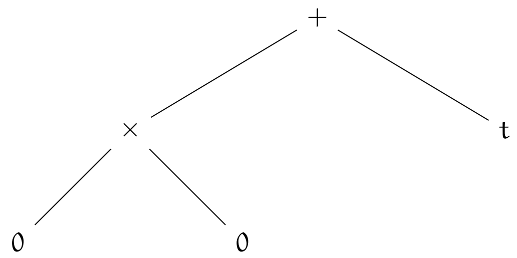
Eksempel

- Den første ukjente termen må begynne med $\times$, så syntakstreet må se ut som



hvor $\mathtt{ttt} = 00 \times +100$. Her kan vi se direkte hva de tre $\mathtt{t}$-ene må stå for, så vi får treet

Eksempel



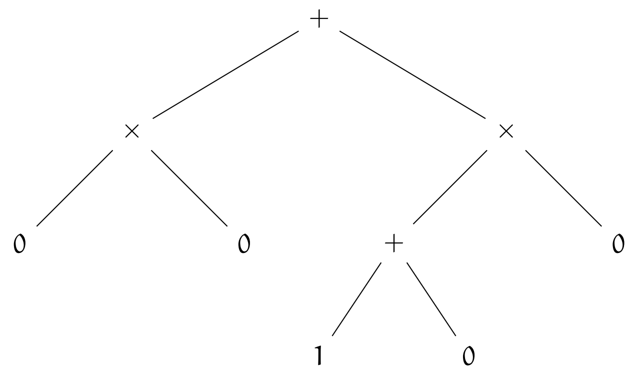
hvor $\mathtt{t} = \times + 100$.

Eksempel

- 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 følgende fullstendige syntakstre.

Eksempel



Skrevet på vanlig infiks-form får vi

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

Mer om notasjon

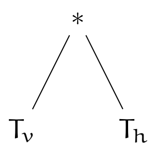
- Når man bruker baklengs polsk notasjon på en lommeregner, taster man inn tall og funksjonsuttrykk som $+$, exp, sin i rekkefølge.
- Hver gang man taster inn et funksjonsuttrykk, vil lommeregneren oppfatte siste tall, eller de to siste tallene, som input, og erstatte disse med funksjonsverdien.
- Det 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.

Mer om notasjon

- Syntakstreet for en term eller et utsagnslogisk uttrykk er uavhengig av om vi har brukt vanlig infiks notasjon, forlengs polsk eller baklengs polsk notasjon.
- Når syntakstreet er gitt, så kan disse uttrykkene gjenskapes ved hjelp av de ulike traverseringsalgoritmene.
- Det er enkelt å lage en algoritme som tar et uttrykk, sjekker om det er korrekt og som eventuelt gir syntakstreet som output.

Induksjon og rekursjon for binære trær

- Mengden av binære trær kan også defineres *induktivt*.
- Utgangspunktet, eller *induksjonstarten*, blir nulltreet, treet som består av en node og ingen kanter.
- Denne noden er da både *rot* og *blad*.
- *Induksjonskrittet* består i at vi tar to binære trær T_v og T_h, en ny rotnode og to nye kanter, mot venstre til roten i T_v og til høyre mot roten i T_h.



Sammensetning av to binære trær til ett.

Induksjon og rekursjon for binære trær

- Vi kan oppfatte denne sammensetningen som en form for *sum* av to binære trær: Vi legger sammen to trær og får et nytt tre.
- Vi kan godt skrive $T_v \oplus T_h$ for denne sammensetningen av trær.
- Merk at den kommutative loven ($x \oplus y = y \oplus x$) ikke gjelder; det er essensielt hvilket av trærne som settes til venstre og hvilket som settes til høyre.
- Som grafer betyr det ikke så mye, men for trerekursjon er det viktig, siden vi der kan referere til venstre og høyre deltre.
- Vi skal nå se på en form for produkt av trær.
- Vi skriver $*$ for nulltreet.

Induksjon og rekursjon for 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 ser at effekten er å erstatte alle bladnodene i T med kopier av S.
- Generelt kan vi definere en funksjon f ved rekursjon over oppbyggingen av binære trær ved følgende.
  1. Bestemme hva $f(*)$ er, når $*$ er nulltreet.
  2. Bestemme hvordan $f(T)$ avhenger av de to deltrærne $f(T_v)$ og $f(T_h)$, når T er et sammensatt tre.

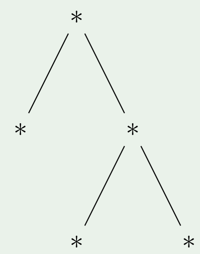
Induksjon og rekursjon for binære trær

Oppgave

Definer treeksponering S^T ved trerekursjon på T ved

1. $S^* = *$
2. $S^T = (S^{T_v} \otimes S) \oplus (S^{T_h} \otimes S)$ når T er et sammensatt tre.

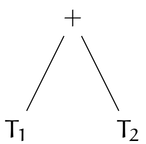
La T og S begge være treet



Finn S^T.

Induksjon og rekursjon for binære trær

- Hvis vi går tilbake til syntakstrær, så kan vi se hvordan de tre ulike notasjonsformene kan defineres via trerekursjon.
- De tre algoritmene svarer til hver sin rekursive funksjon.
- Vi definerer funksjonen $\mathtt{infiks}$ ved trerekursjon på følgende måte.
  - Hvis roten i T er en bladnode med merke a, så lar vi $\mathtt{infiks}(T) = a$.
  - Hvis T er på formen



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

- De to andre funksjonene er definert på samme måte for bladnoder, og på følgende måte for sammensatte trær.
  - $\mathtt{prefiks}(T) = +\mathtt{prefiks}(T_1)\mathtt{prefiks}(T_2)$
  - $\mathtt{postfiks}(T) = \mathtt{postfiks}(T_1)\mathtt{postfiks}(T_2)+$

Bitsekvenser og binære trær

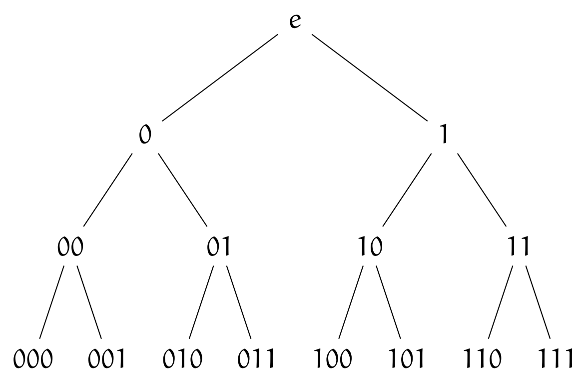
- Det er en intim sammenheng mellom bitsekvenser og binære trær.
- For hver node i et binært tre kan vi lage oss en tilsvarende bitsekvens ved rekursjon på nivået (avstanden til roten) til noden.
- La $\mathtt{bit}(*)$ være den tomme sekvensen hvis $*$ er rotnoden.
- Hvis a er en node med to barn, b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

Bitsekvenser og binære trær

- Omvendt vil en 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.

Bitsekvenser og binære trær

- Hvis vi markere nodene i et binært tre med bitsekvensene, så får vi følgende bilde.



Litt om strømmer

- 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.
- 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.

Litt om strømmer

- Vi tenker oss altså en situasjon hvor prosedyren avslutter med et svar uansett hvilken strøm den fores med.
- For enhver strøm 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 hver enkelt strøm.
- 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?

Litt om strømmer

- 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 (eller begge).
- Start en digital strøm med 0 om venstre deltre er uendelig og med 1 om det er endelig. La T_1 være det tilsvarende uendelige deltreet.
- Fortsett strømmen med 0 om venstre deltre i T_1 er uendelig og med 1 om det er endelig (da er høyre deltre i T_1 uendelig).

Litt om strømmer

- 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.

Litt om strømmer

- Påstanden vi nå har vist, 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.