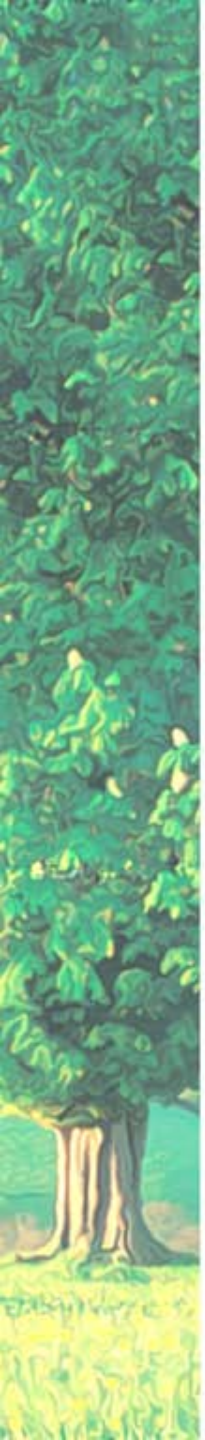
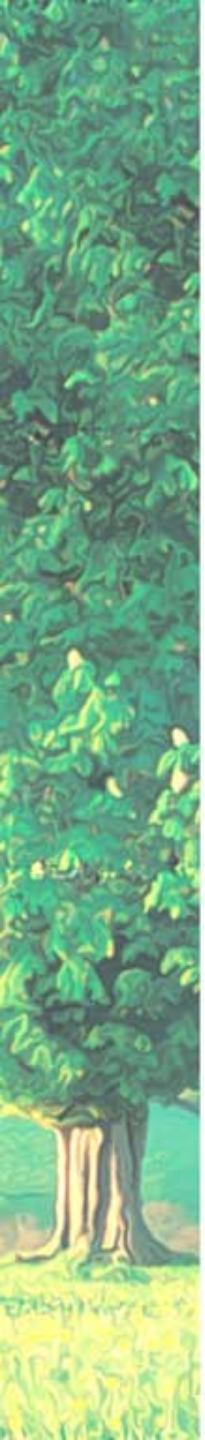


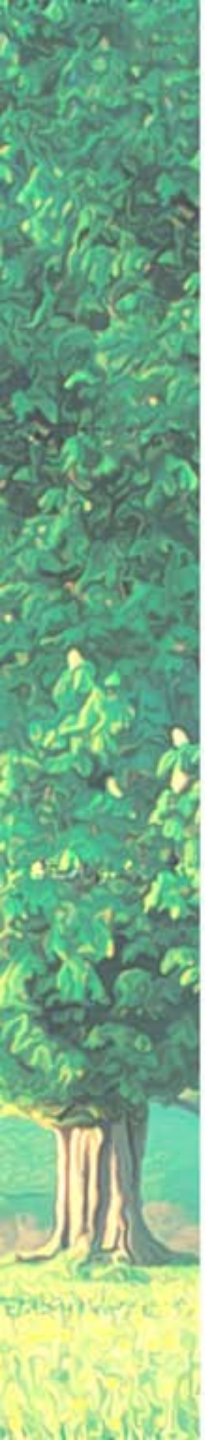
Kap 9 Tre Sist oppdatert 15.03

- Definere et tre som en datastruktur.
- Definere begreper knyttet til tre.
- Diskutere mulige implementasjoner av tre
- Analysere implementasjoner av tre som samlinger.
- Diskutere metoder for å gjennomgå tre.
- Undersøke et eks. med binært tre



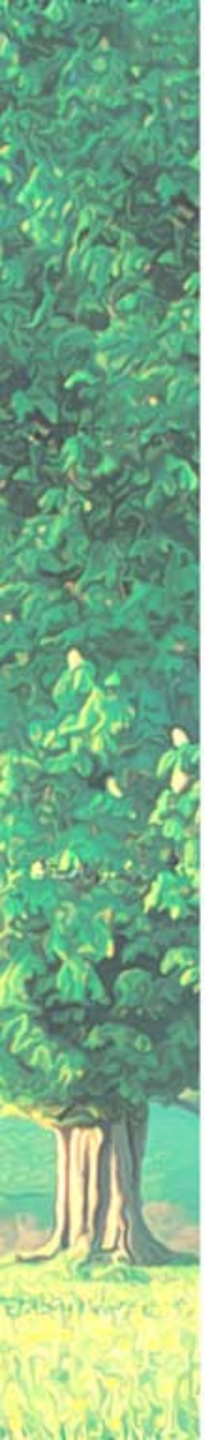
Tre

- Et tre er en sammenhengende graf (en graf = mengde av noder og kanter) uten sykler som oppfyller bestemte krav
- En sti i et tre er en liste av ulike noder der påfølgende noder er forbundet med kanter
- Et tre er en ikke-lineær struktur definert ved at hver node i treet, unntatt den første noden kalt rot, har eksakt en foreldre.
- Operasjoner på tre er avhengig av type tre og bruk.



Definisjoner

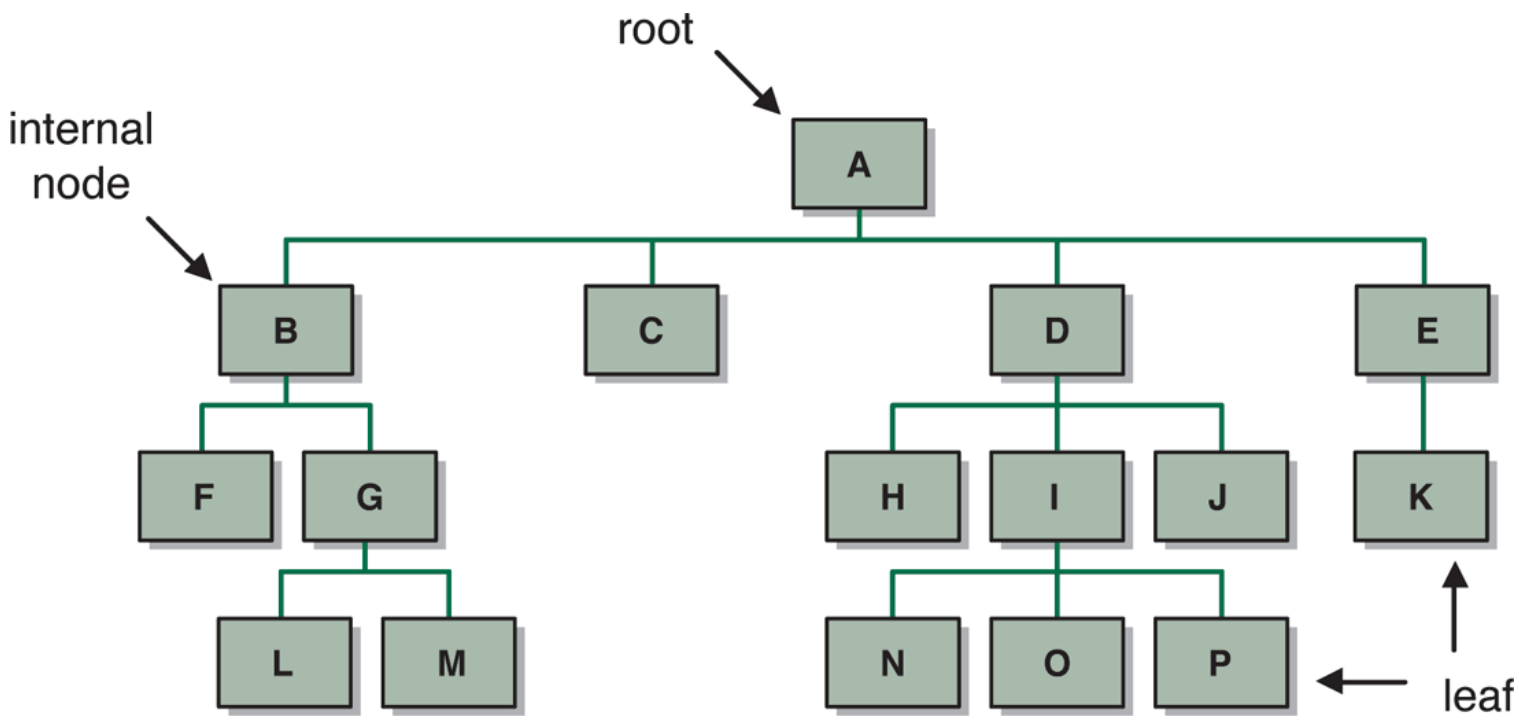
- *En node* referer til en lokasjon i treet hvor et element er lagret.
- I starten av treet fins det en node kalt roten som ikke har en foreldre.

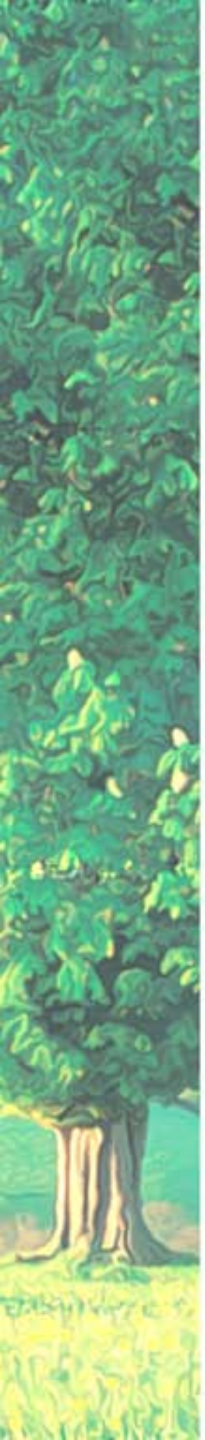


Definisjoner

- Hver node i treet peker på noder kalt *barn*, som er plassert under i trestrukturen (unntatt nodene på siste “nivå”).
- En node kan ha flere barn (søsken).
- En node som ikke har noen barn kalles et *blad.*
- En node som ikke er rot og som har minst et barn er kalt en *intern node.*

FIGUR 9.1 Tre- terminologi



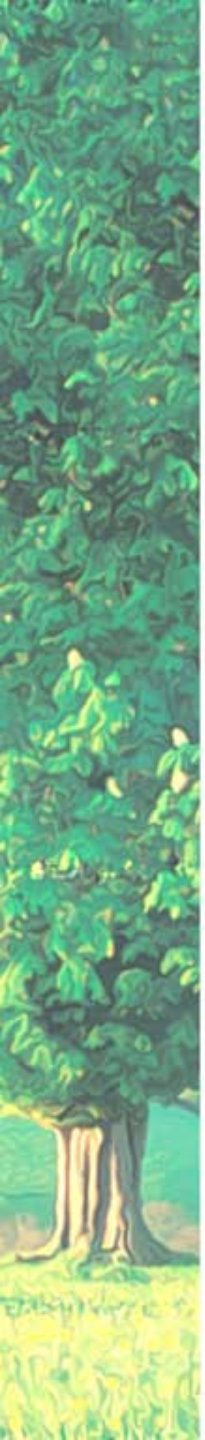


Definisjoner

- Enhver node plassert under en annen node K og på en sti fra K til denne noden er kalt etterfølgeren til K.
- Enver node plassert over en annen node L og på en sti fra roten til L er kalt forgjenger til L.
- Alle barn til samme node er kalt søsken.
- Et tre der hver node har maksimalt N barn er kalt et “N-ary” tre, tre av orden N.
  Et tre der hver node har maksimalt to barn er kalt et ***binært tre.***

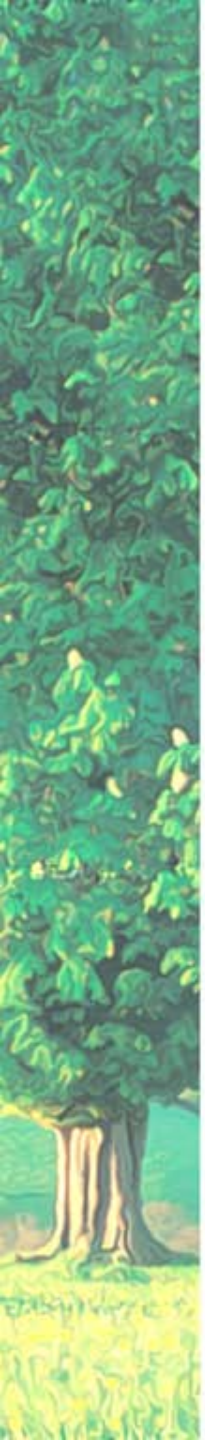
Definisjoner

- Et *undertre* (subtre) til et tre lages ved å velge en node S i treet som rot til undertreet sammen med alle etterfølgere til S og tilhørende kanter. (Kan definere et tre på en rekursiv måte. Se figurer)



Definsjoner

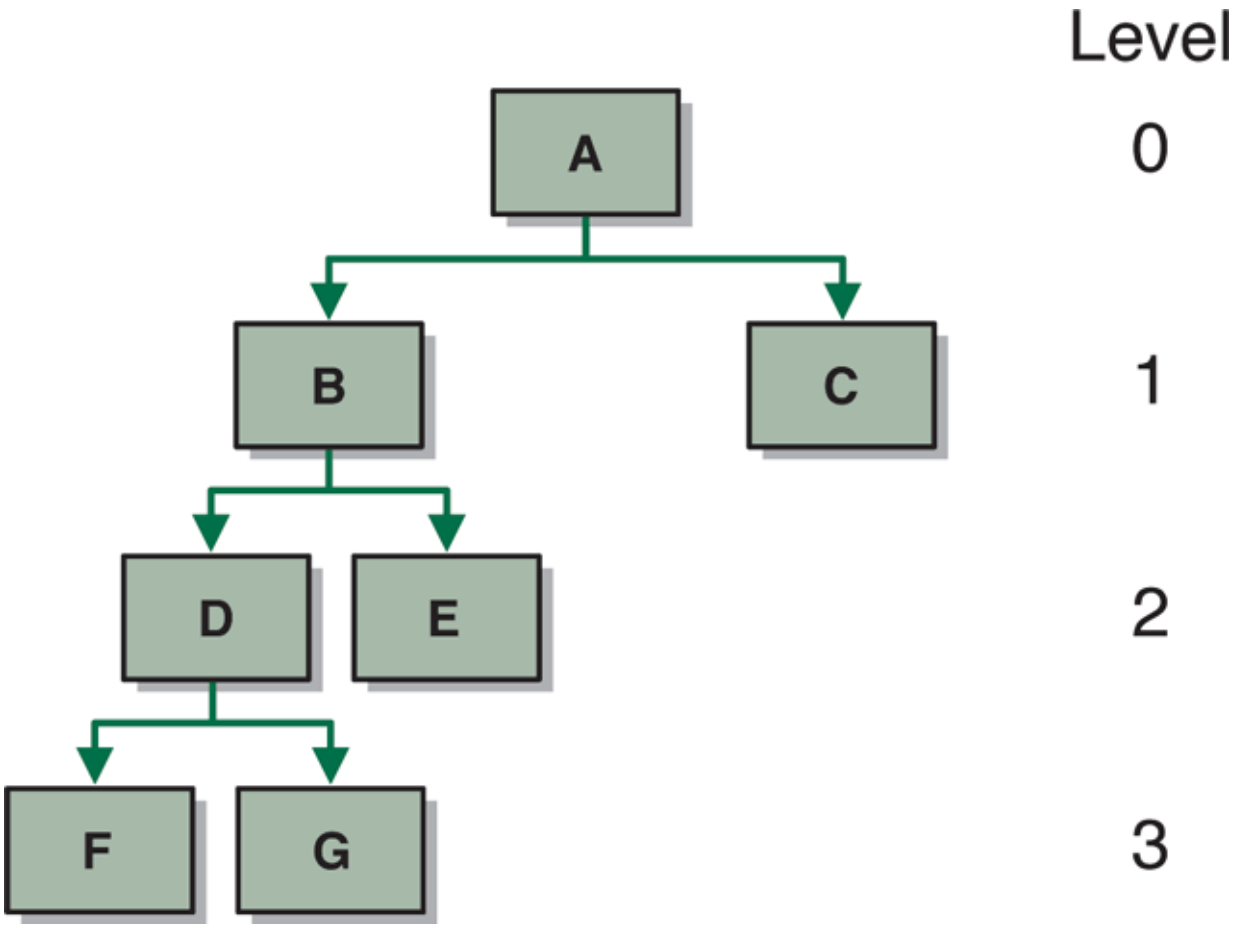
- Hver node i treet er plassert på et bestemt *nivå* eller *dybde* i trestrukturen.
- Nivået til en node er lengden av stien fra roten til noden.
- Denne *stilengden* bestemmes ved å telle antall kanter fra roten til noden.
- Roten er plassert på nivå 0, barn til roten er plassert på nivå 1, barnebarn til roten er plassert på nivå 2 osv.

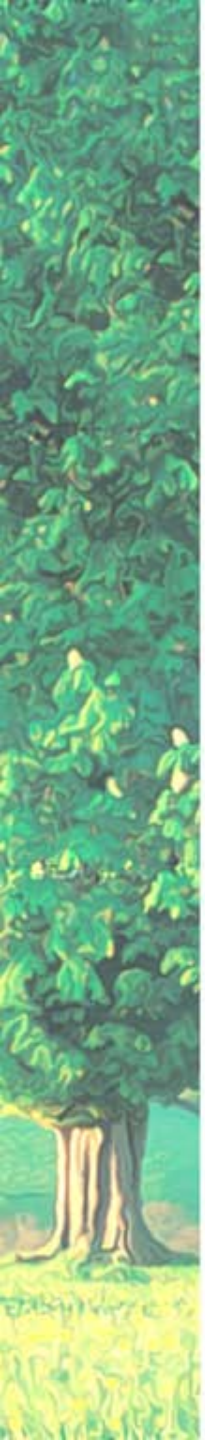


Definisjoner

- *Høyden* til et tre er lengden til den lengste stien fra roten til et blad, altså lik største nivåtallet.
- Høyden til treet på neste lysark er 3.
- Stien fra roten(A) til bladet(F) har lengde 3.
- Stien fra roten(A) til bladet (C) har lengde 1.

FIGUR 9.2 Stilengde og nivå
Eks. på et binært tre





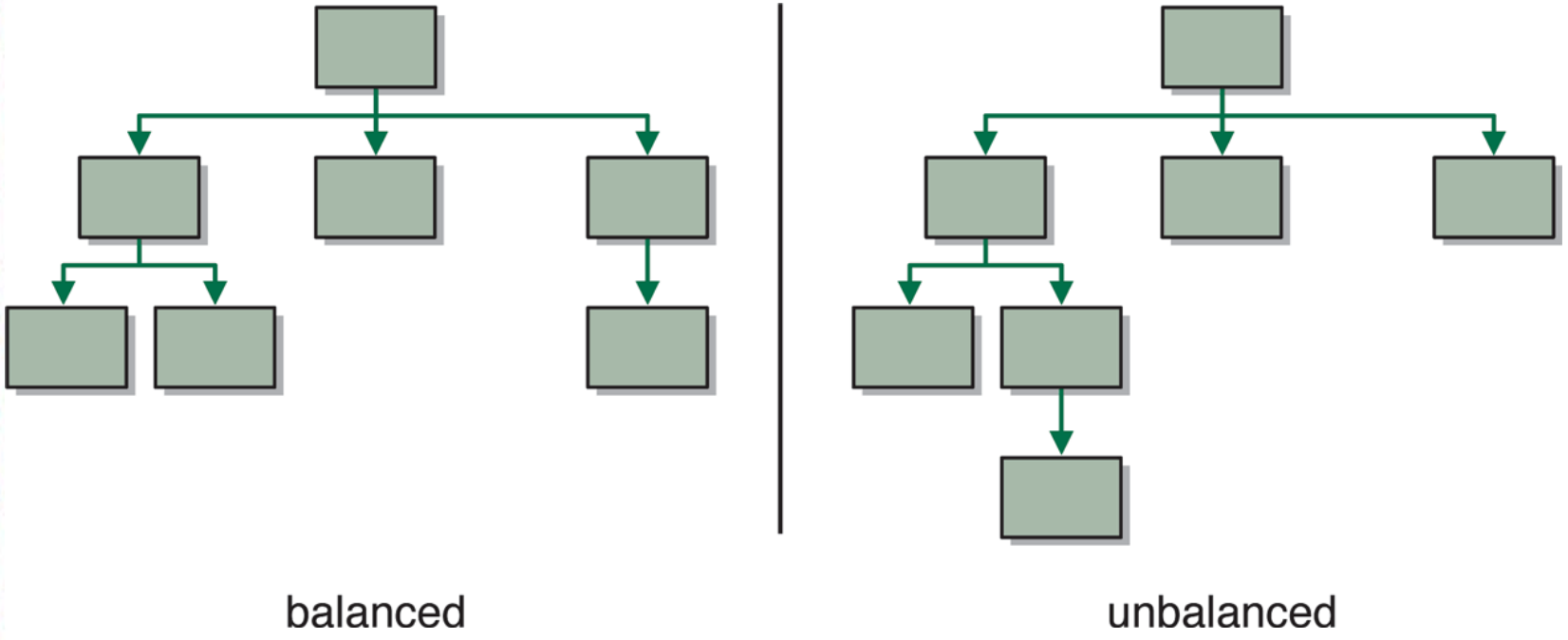
Definisjoner

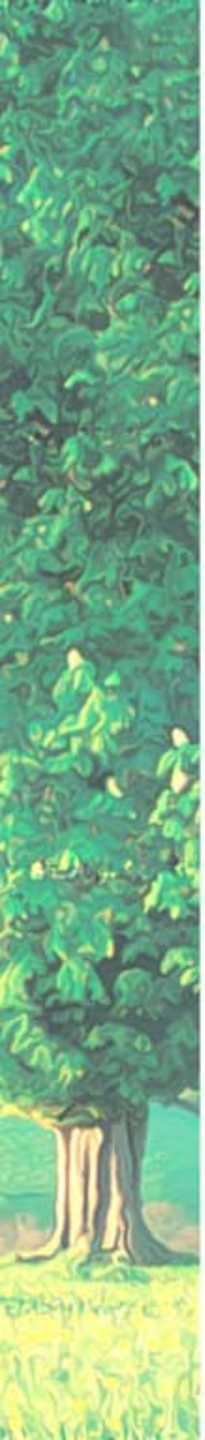
- Et tre er *balansert* hvis alle bladene er plassert på samme nivå eller *tilnærmet* på samme nivå

- Noen algoritmer definerer balansert tre ved at alle bladene er plassert på nivå h eller på nivå h -1 der h er høyden til treet, $\mathbf{h \approx log_N n}$ for et tre av orden N. n er her antall noder.

- For et *balansert binært tre* (binært tre er et tre der hver node har maks to barn) blir høyden $\mathbf{h \approx log_2 n}$ (sml også analyse for binærsøk der vi så på antall halveringer).Maks antall noder på nivå j er $\mathbf{2^j}$.

FIGUR 9.3 Balansert og ubalansert tre

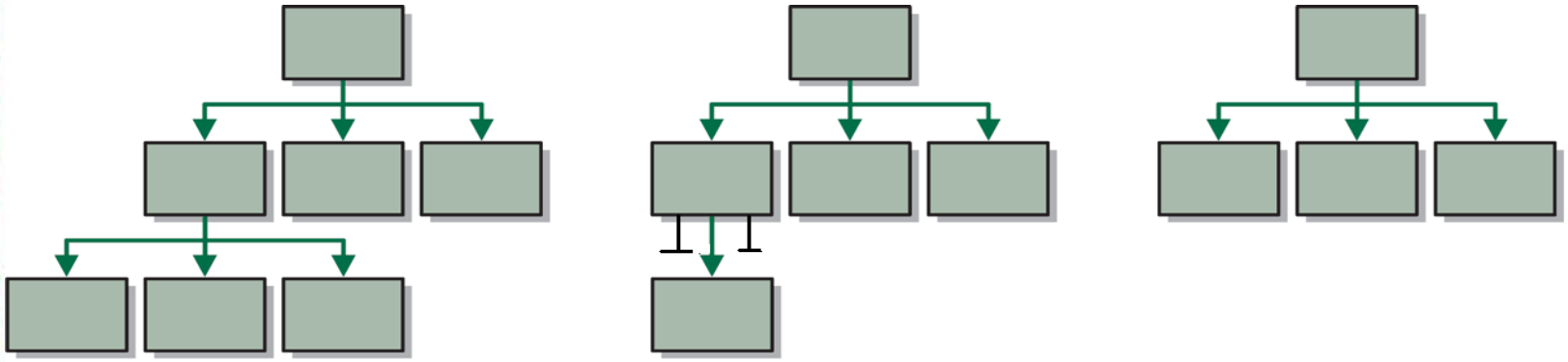


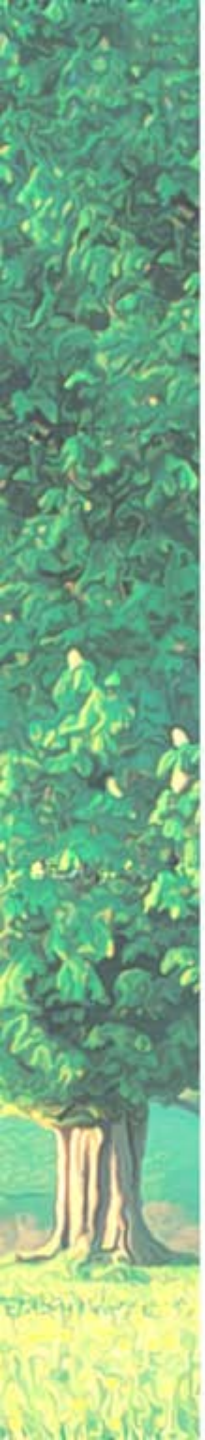


Definisjoner

- Begrepet *fullt* og *komplett* tre er relatert til balansering av et tre.
- Et *fullt tre* er et tre som er fullt på alle nivåer.
- Et *komplett tre* er et tre som enten er fullt eller fullt til siste nivå med blader plassert fra venstre.
- På neste lysark: Tre a er komplett, men ikke fullt. Tre b er ikke komplett (og dermed ikke fullt) Tre c er fullt (og dermed komplett).

FIGUR 9.4 Noen eksempler på komplett tre



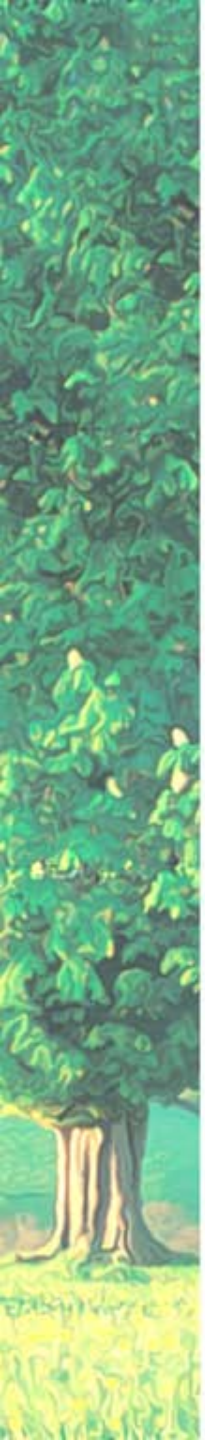


Kjedet representasjon av tre

- Hver node kan defineres som objekt av klasse ***TreNode*** i analogi med klassen LinearNode vi brukte for kjedet liste
- For et binært tre kan vi definere en klasse ***BinærNode*** som på neste lysark.
- NB! Operasjonene på det binære treet er metoder i en større klasse BinærTre som inneholder BinærNode (sml. KjedaListe). Operasjonene definerer vi senere.

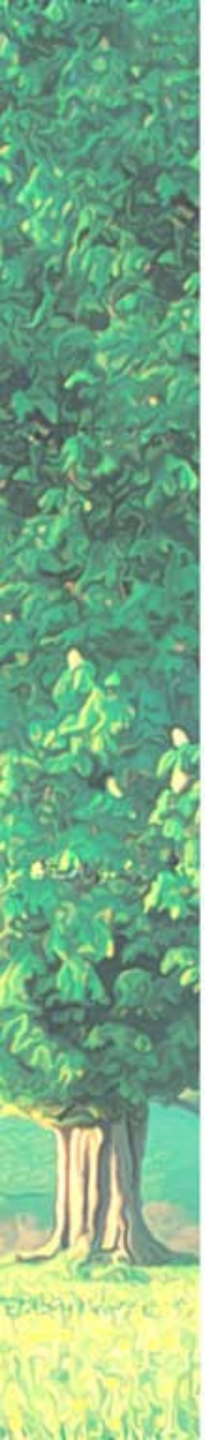
Kjedet implementasjon (2)

```
public class BinærTreNode<T>{
   private T element;
   private BinærTreNode<T> venstre, høyre;
  /*****************************************************
   BinærTreNode (T el) {
      element = el;
      venstre = null;
      høyre   = null;
   }…
}//class
```



Kjedet implementasjon (3)

- Hver node inneholder typisk en referanse til elementet og for hver av de mulige barna.
- I noen implementasjoner er det også nyttig å ha en peker til foreldrenoden (Feks. i såkalte tråklete tre).



Tabellimplementasjon av tre

- For noen typer av tre, spesielt binære tre kan en beregne plassering av noder når en bruker tabellimplementasjon. Vi antar nå et binært tre.

- Anta elementene er lagret påfølgende i en tabell kalt *tabell* der roten lagres som *tabell[0].*

- Bemerk at ikke alle interne noder har maks. antall barn og videre at bladene har ikke noen barn.

Tabellimplementasjon (2)

- Barna til roten lagres påfølgende, først med venstre barn, deretter høyre barn. Tilsvarende for neste nivå. Se neste figur. (i er posisjonen eller indeksen i tabellen)

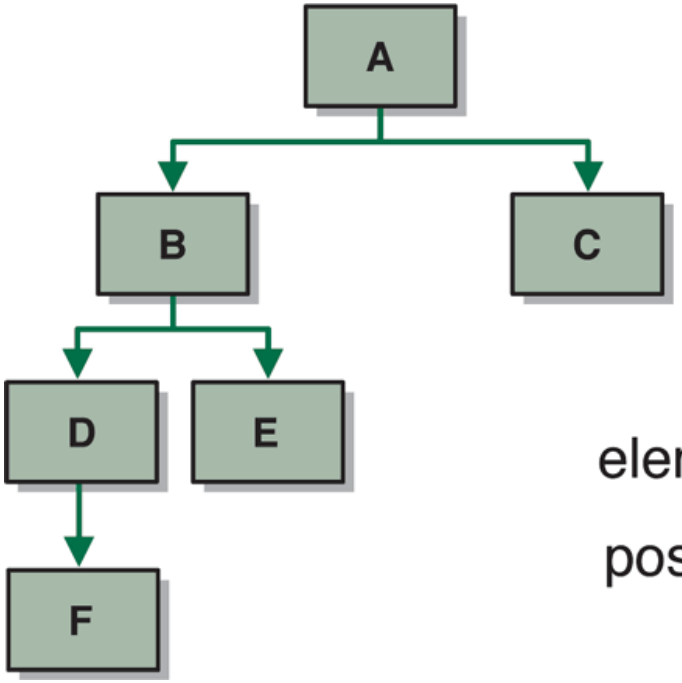
- For elementet tabell[i] finner vi eventuelt venstre barn som tabell[2*i+1] og eventuelt høyre barn som tabell[2*(i+1)].

Tabellimplementasjon (3)

- Foreldren til elementet tabell[i] er gitt ved tabell[(i-1)/2] enten elementet er venstrebarn eller høyrebarn.

- Dersom treet ikke er komplett eller “lite komplett”, så vil vi få mange tomme plasser i tabellen.

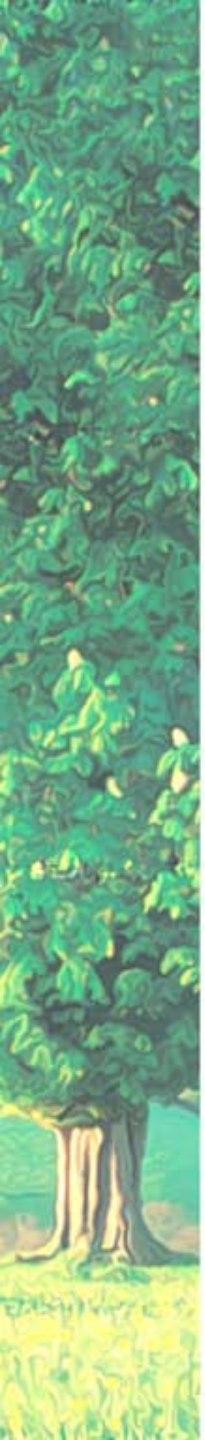
FIGURE 8.5 Beregning av posisjon av foreldre og barn



element	A	B	C	D	E			F
position	0	1	2	3	4	5	6	7

Tabellimplementasjon (4)

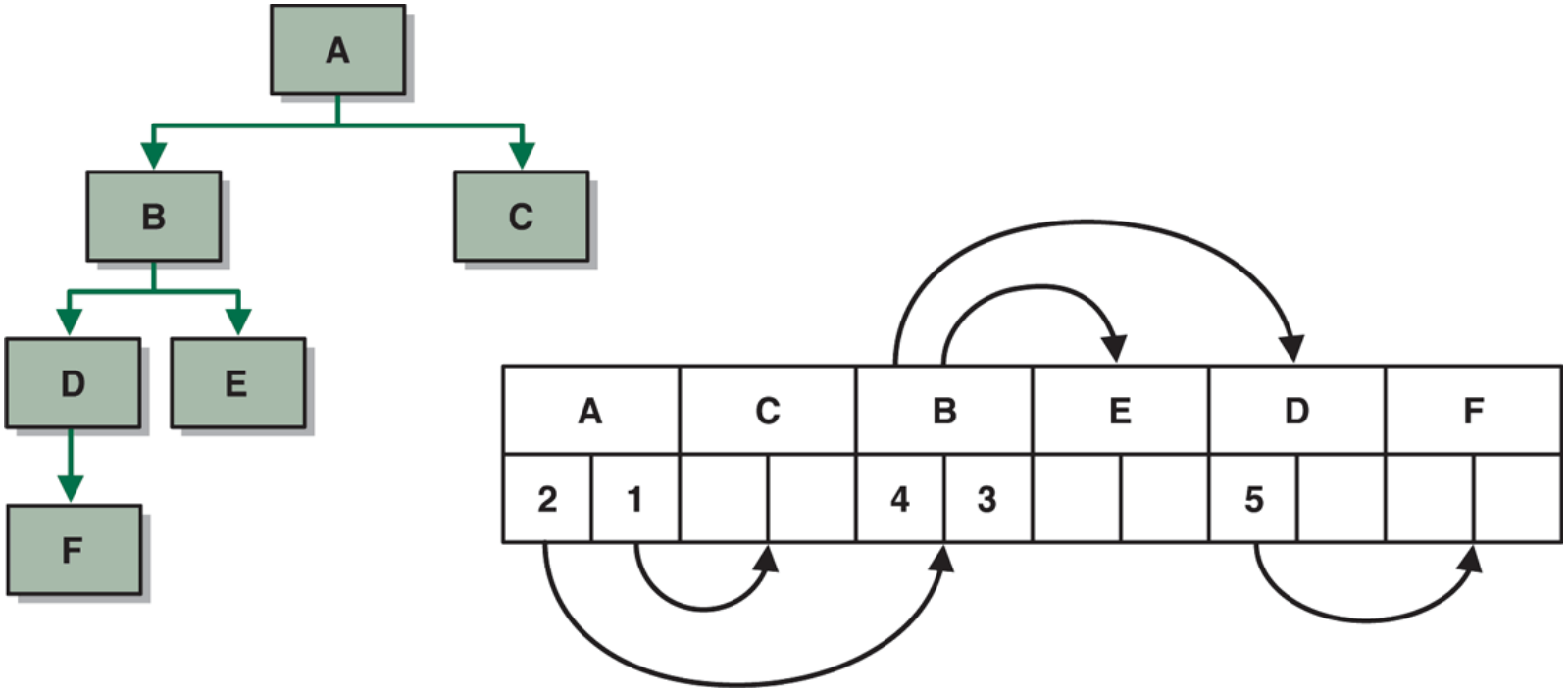
- En annen variant av tabellimplementasjon er modellert etter måten operativsystemet administrerer minne.
- I stedet for å tildele elementer av treet til tabellposisjoner ved lokasjon i treet, blir tabellposisjonene tildelt kontinuerlig etter FIFO- prinsipp.



Tabellimplementasjon (5)

- Hver node lagrer tabellindekser til hvert barn i stedet for referanser til barna.
- **Fordelen er at elementene lagres kontinuerlig** og at vi ikke får unødvendig sløsing av plasser.
- Men det er mer komplisert ved bl.a. sletting av et element siden resten av elementene da må skiftes for at vi skal ha kontinuerlig lagring. Fig.9.6.

FIGUR 9.6 Simulert kjedet strategi for tabellimplementasjon av tre



Analyse av tre

- Antall Noder, n i et fullt binært tre med nivåer 0..k er gitt ved :

$$n = \sum_{i}^{k} 2^i = 2^0 + 2^1 + ... + 2^k = 2^{k+1} - 1$$

(geometrisk rekke)

Tegn opp!

Analyse av tre

- La n være antall noder i et fullt binært tre.
- Høyden h til et til et fullt binært tre eller til et komplett binært tre med nivåer 0..k er gitt ved: h = k (høyden er det største nivåtallet).

Fra forrige formel:

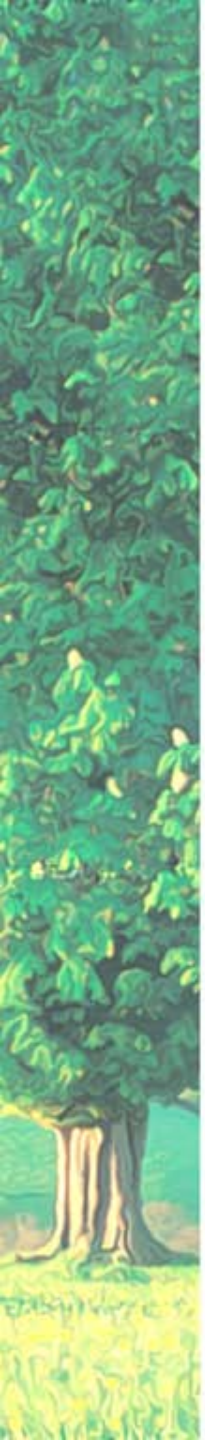
$$n = 2^{k+1}-1 = 2^{h+1}-1$$

Tar logaritmen og får:

$$h+1 = \log_2(n+1)$$

$$h = \lfloor \log_2 n \rfloor + 1.$$

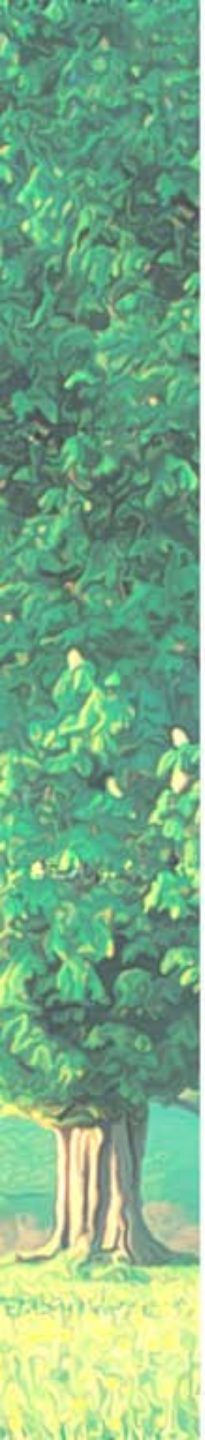
Dvs O(logn)



Analyse av tre

- Tre er en nyttig og effektiv måte å implementere samlinger på.

- I vår analyse av listeimplementasjoner i kap.6 fant vi at forventet antall sml. ved finnoperasjonen var ≈ n/2, dvs. O(n) der n er antall elementer.



Analyse av tre

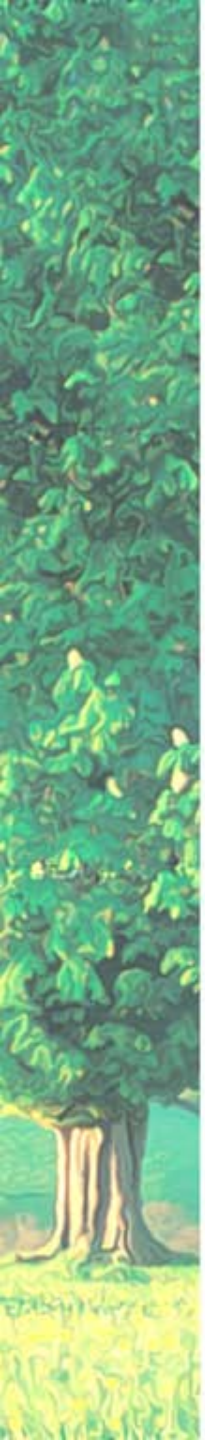
- Hvis vi implementer en ordnet liste ved å bruke et **balansert binært tre** med tilleggs-egenskapen at venstre barn alltid er mindre enn foreldren og videre at foreldren er mindre eller lik høyrebarn, så kan vi **forbedre effektiviteten** av finn-operasjonen til **O(logn)**
- Et slikt tre kalles et ***binært søketre*** **(Kap.10).**
- Begrunnelsen for resultatet over er at **høyden** av et slikt **balansert binært** tre alltid vil være ≈ $\log_2 n$ når n er antall elementer i treet.

Analyse av tre

- For et balansert tre av orden N med n elementer vil høyden være **≈ $\log_N n$.**

  Tegn et balansert tre av orden N lik 4 og n = 21.
  Hva blir høyden? Stemmer formelen?
  Lag et tre av orden N lik 2 (binært tre) og n = 21.
  Hva blir nå høyden? Stemmer formelen?

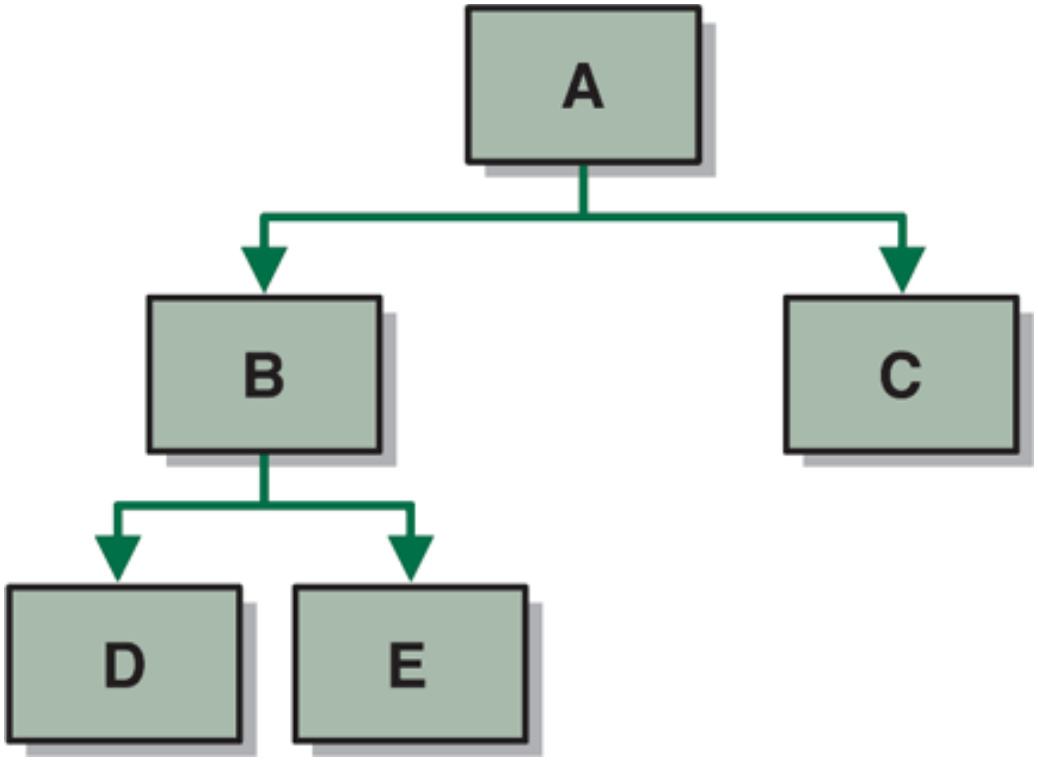
- For **binære søketre (Kap.10)** vil vi i verste tilfelle være garantert å **søke i en sti fra roten til et blad.**

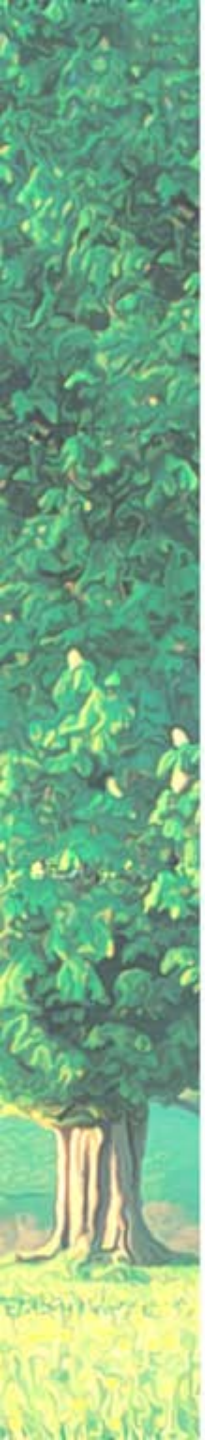


Gjennomgang av tre

- Det er fire hovedalgoritmer for gjennomgang av tre.
  - Preorden gjennomgang
  - Inorden gjennomgang
  - Postorden gjennomgang
  - Nivåorden gjennomgang
- Vi starter alltid en gjennomgang av tre med roten.

FIGUR 9.7 Et binært tre



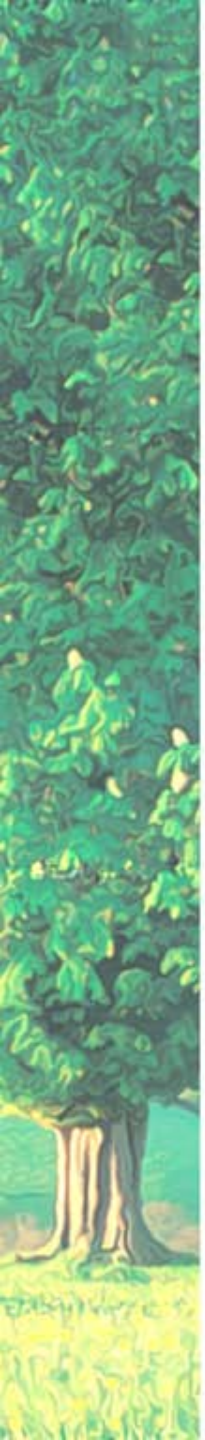


Preorden gjennomgang

- **Preorden er utført ved først å besøke hver node, deretter besøke venstre undertre i preorden og så høyre undertre i preorden.**

- **En preorden gjennomgang er:**

  A B D E C



Preorden gjennomgang

- **Prinsipp for preorden gjennomgang av et binært tre:**

  besøk noden

  gjennomgå venstre undertre i preorden

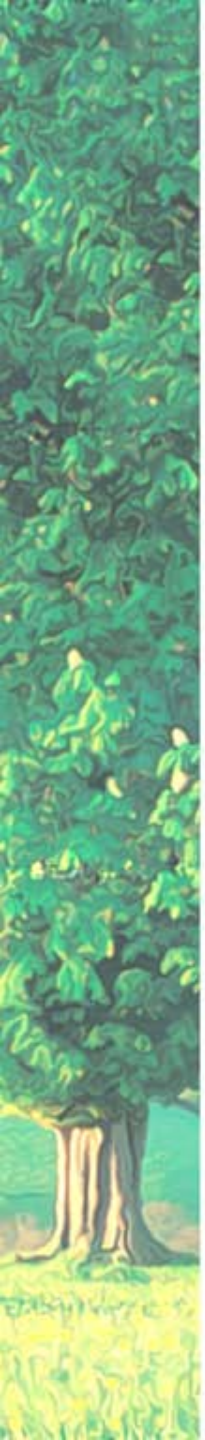
  gjennomgå høyre undertre i preorden

Inorden gjennomgang

- **Inorden gjennomgang er utført ved først å besøke venstre undertre i inorden, deretter besøke noden og så besøke høyre undertre i inorden**

- **En inorden gjennomgang av forrige tre gir:**

  D B E A C



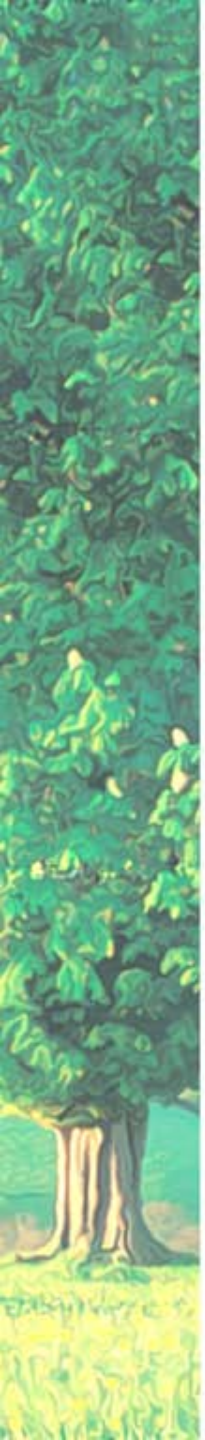
Inorden gjennomgang

- **Prinsipp for inorden gjennomgang av et binært tre:**

  gjennomgå venstre undertre i inorden

  besøk noden

  gjennomgå høyre undertre i inorden

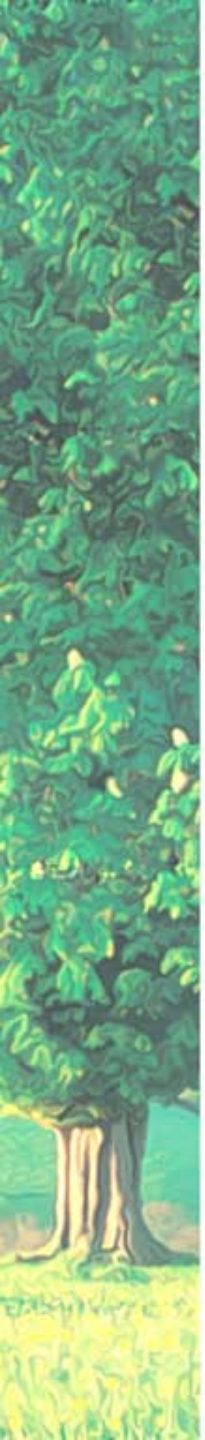


Postorden gjennomgang

- **Postorden gjennomgang er utført ved først å gjennomgå venstre undertre i postorden, deretter gjennomgå høyre undertre i postorden og så besøke noden.**

- **Gitt det samme treet som før.**
  En postorden gjennomgang er:

  D E B C A



Postorden gjennomgang

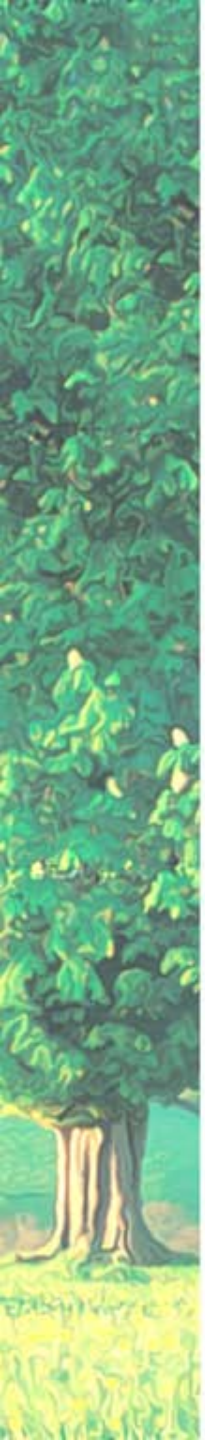
- **Prinsipp for postorden gjennomgang av et binært tre:**

gjennomgå venstre undertre i postorden
gjennomgå høyre undertre i postorden
besøk noden

Nivågjennomgang

- **Nivågjennomgang er utført ved å besøke alle nodene på hvert nivå (søskene) fra venstre mot høyre, et nivå om gangen.**

- **Gitt det samme treet.**
  En nivå gjennomgang er:
  A B C D E



Nivå-gjennomgang alg.

Opprett en kø

Opprett en tom liste

Legg roten inn i køen

så lenge køen er ikke-tom **utfør** {

ta ut et element fra køen

hvis elementet er ikke-null **så**

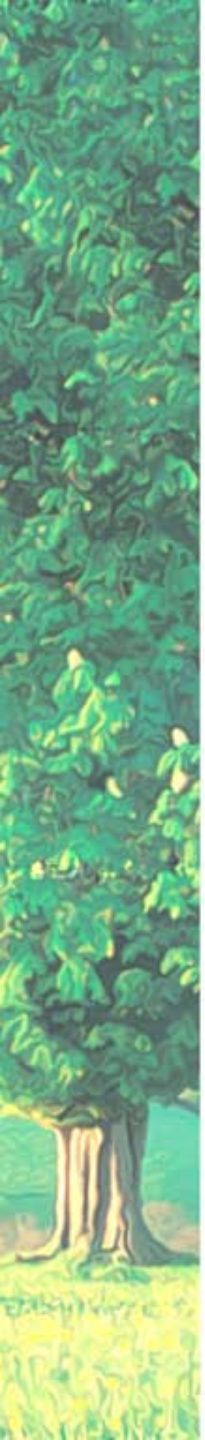
legg elementet bak i listen
legg barna til elementet i køen

ellers

legg null i listen

}

Elementene i listen ligger nå i nivårekkefølge.



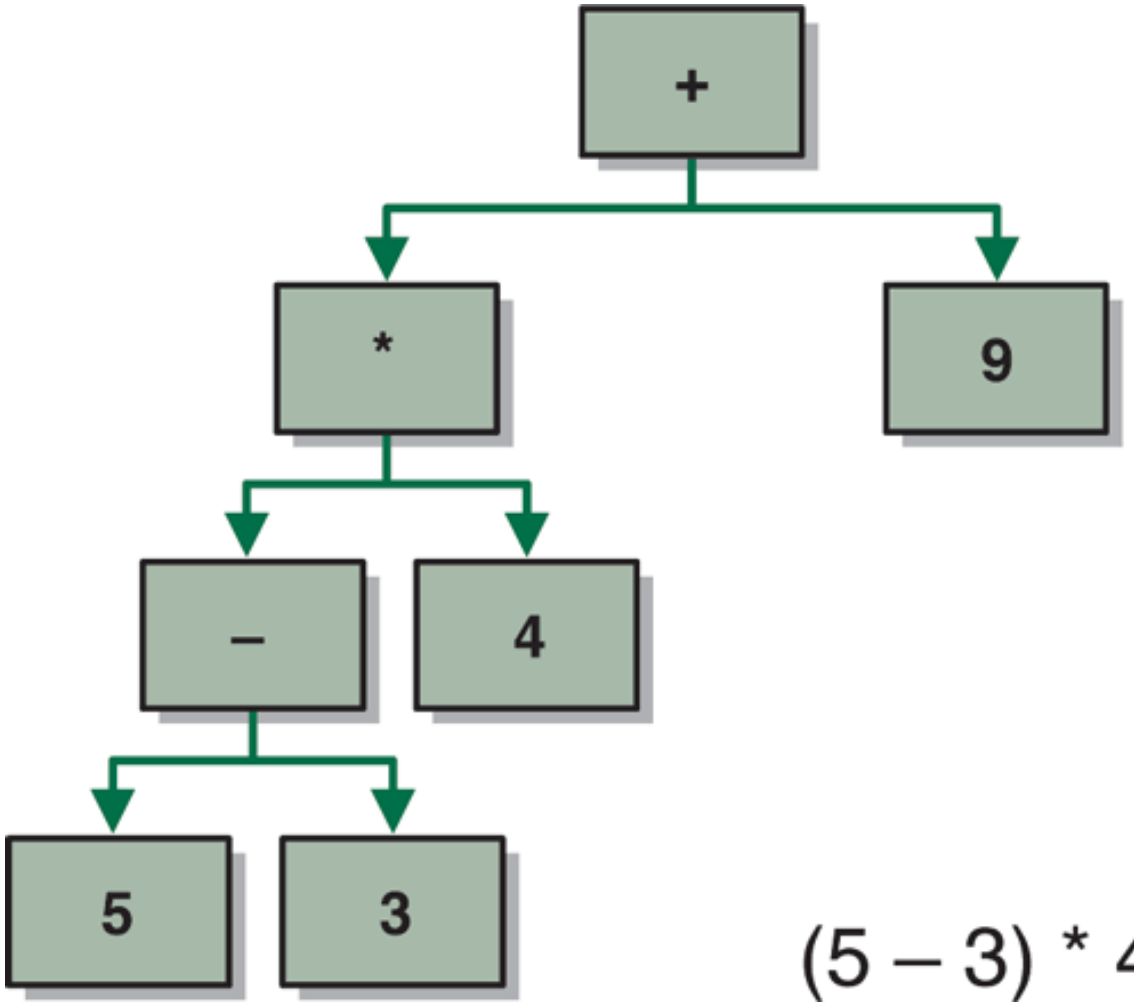
Implementere et binært tre

- Vi kan se på en felles mengde av operasjoner for binære tre:

  - **fjernVenstreUnderTre**
  - **fjernHøyreUnderTre**
  - **fjernAlleElementer**
  - **erTom**
  - **antall**
  - **inneholder**
  - **finn**
  - **toString**

- Vi kan ikke definere en leggtil-operasjon før vi har tatt stilling til hvordan treet skal brukes.

FIGUR 9.10
Eksempel på et tre for aritmetisk uttrykk med binære operatorer



$(5 - 3) * 4 + 9$