Föreläsning 10

Träd

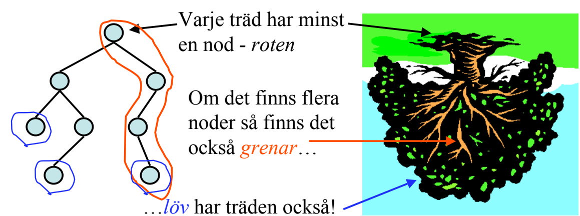
Innehåll

- Modeller/tillämpningar för träd
- Organisation och terminologi
- Signaturdiagram för ordnat träd
- Olika typer av träd
- Trädalgoritmer
- Implementation av träd

Modeller/tillämpningar för träd

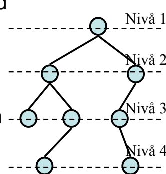
- Modell:
  - Ordnat träd:
    - Ordervägarna i ett regemente
  - Binärt träd:
    - Stamtavla/släktträd
- Tillämpningsexempel inom datavärlden:
  - Filsystem
  - Klasshierarkier i Java/OOP
  - Besluts-/sök-/spelträd inom AI
  - Prologs exekvering

Organisation och terminologi (1)



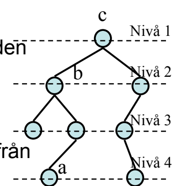
Organisation och terminologi (2)

- Elementen i ett träd kallas för *noder*. En nod har en *position* och ev. ett *värde*.
- Värdet på en nod kallas *etikett.*
- Alla noder på en nivå kallas *syskon.*
- Ett träd har ett ändligt antal noder och är en homogen datatyp.
- Föräldra-barn hierarki.
- Delträd = en nod och dess avkomma.



Organisation och terminologi (3)

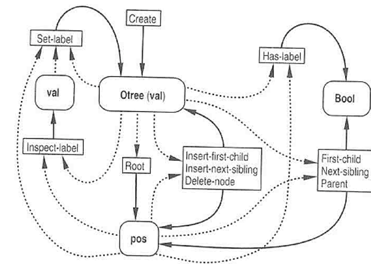
- Ett träds noder finns på olika *nivåer.*
- *Höjden* $h(x)$ för nod x är antalet bågar på den längsta grenen i det träd där x är rot.
  - Höjden av ett träd T, $h(T) = h(\text{roten})$
  - $h(a) = 0$, $h(b) = 2$ och $h(c) = 3 = h(T)$
- *Djupet* $d(x)$ hos en nod x är antalet bågar från x upp till roten.
  - $d(a) = 3$, $d(b) = 1$, $d(c) = 0$
  - $nivå(x) = d(x) + 1$



Signaturdiagram för ordnat träd

- *Navigeringsorienterad* vs *delträdsorienterad* specifikation av träd.
- Om man arbetar med enstaka träd som förändras långsamt löv för löv så är navigeringsorienterad bättre.
- Håller man på med träd och delträd som man vill dela upp eller slå samman är delträdsorienterad bättre.

Signaturdiagram för ordnat träd



Olika typer av träd

- Ordnat träd (ex militärhierarki)
  - Syskonen är linjärt ordnade
- Oordnat träd (ex filsystemet på en dator)
  - Ordningen bland syskonen har ingen betydelse
- Urträd
  - Mer abstrakt än de två förra. Har en egen datatyp som hanterar syskonen.

Olika typer av träd

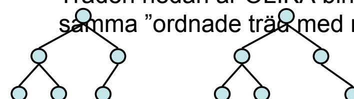
- Riktade träd
  - Kan bara gå i en riktning i trädet.
  - I ett nedåtriktat träd saknas `Parent`.
  - I ett uppåtriktat träd saknas `Children`, måste gå att nå något annat än roten, tex en operation som ger alla löv.
- Binära träd (tex stamtavla)
  - Varje nod har högst två barn

”Begreppspaus”- Om ordning

- Ordnad
  - Används för att beskriva olika sätt att ordna element i ett objekt i en datatyp
- Riktad
  - När det finns en asymmetri när det gäller operationer för att hitta från ett element till ett annat.
- Sorterad
  - När elementvärdena är sorterade enligt någon ordningsrelation

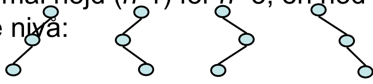
Binära träd (1)

- En nod i ett binärt träd kan ha högst två barn
  - Barnen kallas vänster- och högerbarn.
  - Det andra barnet kan komma före det första
  - Träden nedan är OLIKA binära träd (men samma ”ordnade träd med max två barn”).

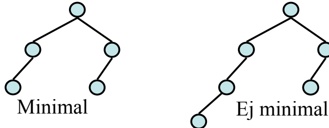


Maximal och minimal höjd

- Maximal höjd (n-1) för n=3, en nod på varje nivå:



- Minimal höjd, man kan inte flytta om några noder och få en mindre höjd:



Binära träd (2)

- För binära träd T med n noder och höjd h gäller:
  - $h \leq n-1$ (maximala höjden)
  - $h \geq \log_2(n+1)-1$
    - Antalet noder på djup i är 2^i dvs 1, 2, 4, 8…
    - Antalet noder totalt i trädet: $n \leq 2^{(h+1)} - 1$
  - Ett träd har minimal höjd om $n > 2^h - 1$ vilket ger
    - $\log_2(n+1)-1 \leq h < \log_2(n+1)$
    - dvs h är av $O(\log_2(n))$

Binära träd (3)

- Man vill ha så grunda träd som möjligt
  - Om vänster och höger delträd är ungefär lika stora har trädet *balans* och vägen till en slumpvis vald nod är $O(\log_2(n))$
- *Komplett binärt träd* (Rätt bra balans)
  - Fyller på trädet från vänster till höger, en nivå i taget.
- *Fullt binärt träd* (Ofta dålig balans)
  - Varje nod är antingen ett löv eller har två barn.

Algoritmer

- Basalgoritmer för träd
  - Djup
  - Höjd
  - Slå ihop
  - Dela upp
  - Beräkna
  - Traversera

Traversering av träd

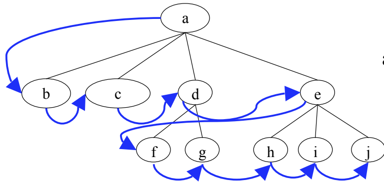
- Tillämpningar av träd involverar ofta att man
  - *Söker* efter ett element med vissa egenskaper
  - *Transformerar* strukturen till en annan struktur
    - Exempelvis sortering och balansering
  - *Filtrerar* ut element med vissa egenskaper
- Alla dessa bygger på att man traverserar strukturen.

Traversering av träd Bredden-först

- Man undersöker en nivå i taget. Först roten, sedan rotens barn, dess barnbarn osv.
  - Kö ofta hjälp vid implementationen.
  - Varje nod i trädet besöks endast en gång, dvs O(n).
  - Söker man något som finns hittar man det.

Traversering av träd-Bredden först

```
Algoritm bfOrder(Tree T)
    input: A tree T to be traversed
for each level L of T do
    for each node of L do
        compute(node)
```



Ordningen:
a, b, c, d, e, f, g, h, i, j

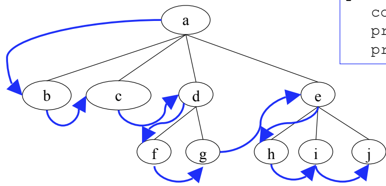
Traversering av träd Djupet-först

- Man följer varje gren i trädet utifrån roten till lövet
  - Stack till hjälp för implementeringen
  - Varje nod besöks endast en gång, dvs O(*n*).
  - Tre varianter:
    - Preorder
    - Postorder
    - Inorder

Traversering av träd - Preorder

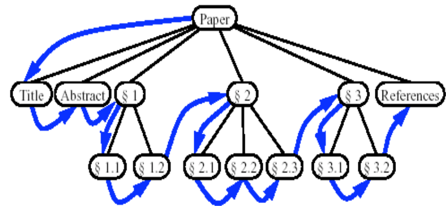
```
Algoritm preOrder(Tree T)
    input: A tree T to be traversed
compute(root(T)) // Do something with node
for each child w of root(T) do
    preOrder(w)
```

```
preOrder(BinTree T)
    compute(root(T))
    preOrder(leftChild(T))
    preOrder(rightChild(T))
```



Ordningen:
a, b, c, d, f, g, e, h, i, j

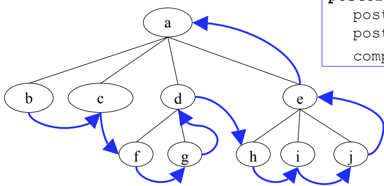
Preorder – Läsa ett dokument…



Traversering av träd - Postorder

```
Algoritm postOrder(Tree T)
    input: A tree T to be traversed
for each child w of root(T) do
    postOrder(w)
compute(root(T)) // Do something with node
```

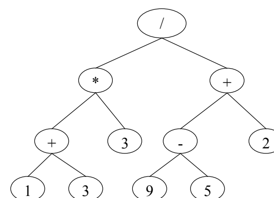
```
postOrder(BinTree T)
    postOrder(leftChild(T))
    postOrder(rightChild(T))
    compute(root(T))
```



Ordningen:
b, c, f, g, d, h, i, j, e, a

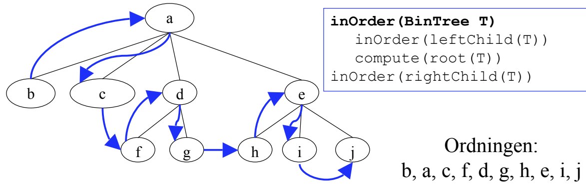
```
Algoritm evaluateExpression(Tree t)
If isLeaf(t)
        return getValue(t)
else
        op ← getValue(t)
        x ← evaluateExpression(leftChild(t))
        y ← evaluateExpression(rightChild(t))
        return x op y
```

Postorder – Beräkna aritmetiska uttryck



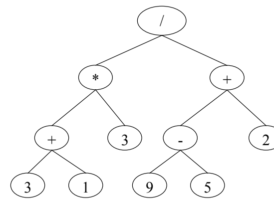
Traversering av träd – Inorder

```
Algoritm inOrder(Tree T)
    input: A tree T to be traversed
node ← root(T)
inOrder(firstChild(T))
compute(node) // Do something with node
for each child w of node (except first) do
    inOrder(w)
```

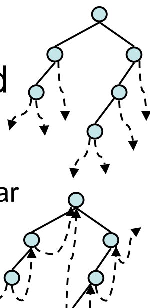


Inorder – Skriva aritmetiska uttryck

```
Algoritm printExpression(Tree t)
    print "("
    if hasLeftChild(t) then
        printExpression(leftChild(t))
    print getValue(t)
    if hasRightChild(t) then
        printExpression(rightChild(t))
    print ")"
```



Trädda binära träd



- Nedåtriktade binära träd har "lediga" länkar.
- Utnyttja dessa för att "trä" genvägar i trädet.
- Det är vanligt att skapa inorder-trädda träd.
- Detta gör att man kan traversera med hjälp av iteration istället för rekursion.
  - Sparar minne

Tillämpningar

- Konstruktioner av andra typer (speciellt binära träd)
- Sökträd
  - Varje nod symboliserar ett givet tillstånd.
  - Barnen symboliserar de olika tillstånd man kan hamna i utifrån förälderns tillstånd.
  - Det gäller att hitta målnoden, dvs ett tillstånd som löser problemet.
  - Inte rimligt att bygga upp alla noder (möjliga) tillstånd.
  - Ofta används heuristik

Tillämpningar

- Planträd och OCH/ELLER-träd
  - Noderna symboliserar hur man bryter ned ett stort problem i mindre delar och i vilken ordning man bör lösa dessa mindre delproblem.
  - Ofta använder man OCH/ELLER-träd där man kan ha OCH-kanter eller ELLER-kanter mellan förälder och barn.
    - OCH – alla barn behövs för lösningen
    - ELLER – något barn behövs

Implementationer av träd

- Oordnat uppåtriktat träd som fält
  - Varje element i en vektor består av ett par: nodens etikett och en referens till föräldern.
    + Tar liten plats
    - Inget bra stöd för traversering (t ex svårt veta vilka noder som är löv)
    - Maximala storleken på trädet måste bestämmas i förväg

Implementationer av träd

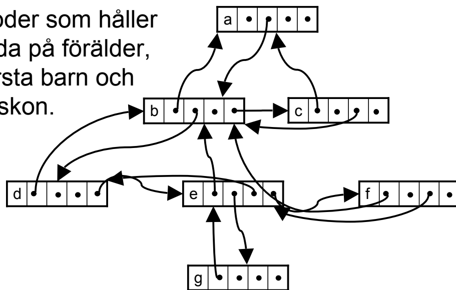
- Ordnat träd som n-länkad struktur
  - Noden i trädet består av n-celler med etikett, länk till föräldern och n-1 länkar till barnen
    + Antalet noder i trädet dynamiskt.
    - Maximala antalet barn bestämt i förväg.
    - Om det är stor variation i antalet barn så finns
      outnyttjade länkar.

Implementationer av träd

- Nedåtriktat ordnat träd som 1-länkad struktur med lista av barn
  - Noden i trädet består av 1-celler med etikett, en länk till en barnlista
    + Antalet noder i trädet dynamiskt.
    + Antalet barn i nätet dynamiskt
- Utöka till 2-celler så blir trädet oriktat
  - Noden får en länk till föräldern, etikett samt länk till barnlista.

Implementationer av träd

- Noder som håller reda på förälder, första barn och syskon.



Implementationer av träd

- Uppåtriktat binärt träd med hjälp av 1-cell
- Nedåtriktat binärt träd med 2-cell
- Oriktat binärt träd med 3-cell

Föräldr alänk	Etikett

Vänster -barn	Etikett	Höger-barn

Föräldra -länk	Vänster-barn	Etikett	Höger barn

Implementationer av träd

- Binärt träd som fält
  - Roten har index 1 och noden med index *i* har
    - sitt vänsterbarn i noden med index 2**i*
    - sitt högerbarn i noden med index 2**i* +1
    - sin förälder i noden med index floor(*i*/2)
  + Tar inget utrymme för strukturinformation
  - Trädet har ett maxdjup (statiskt fält)
  - Krävs ”markörer” för null och tom nod
  - Ev. slöseri med utrymme