F09 - Träd

5DV149 Datastrukturer och algoritmer
Kapitel 9–10

Niclas Börlin
niclas.borlin@cs.umu.se
Anna Jonsson
aj@cs.umu.se

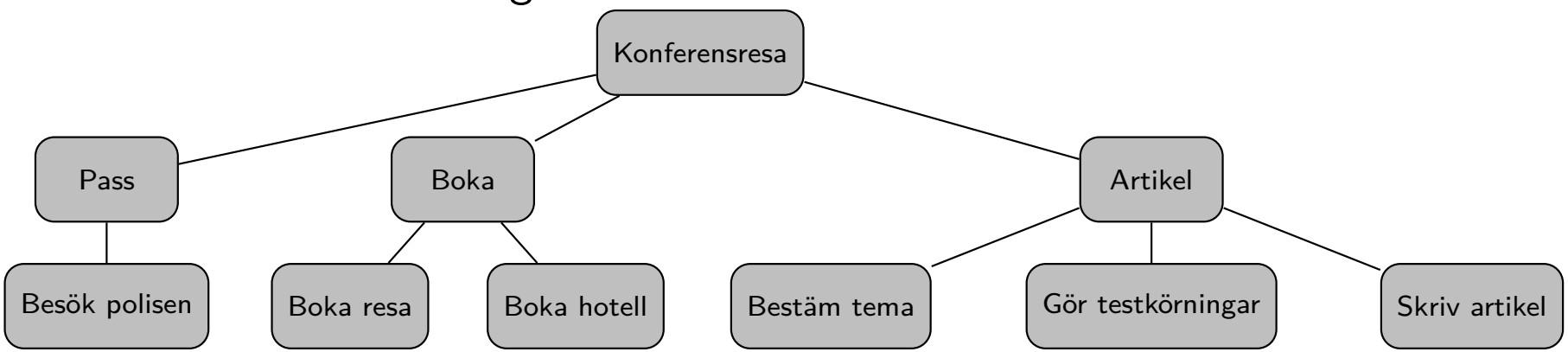
2020-02-17 Mon

Innehåll

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

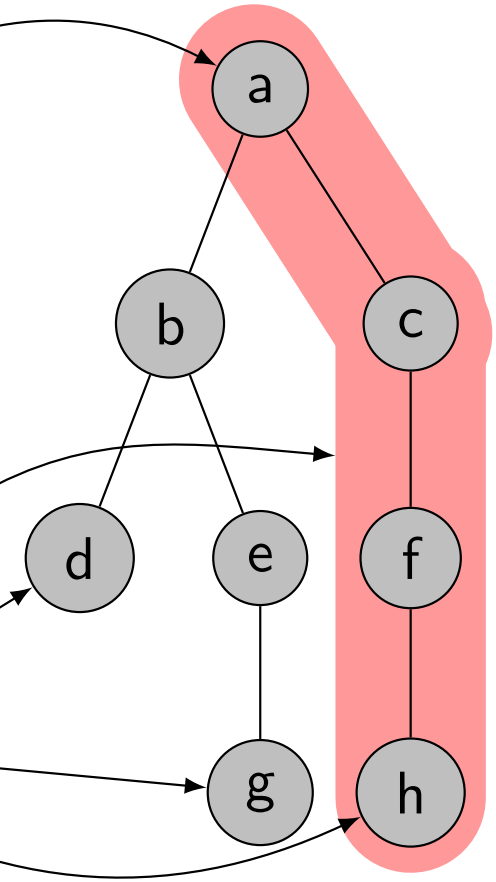
Modeller och tillämpningar

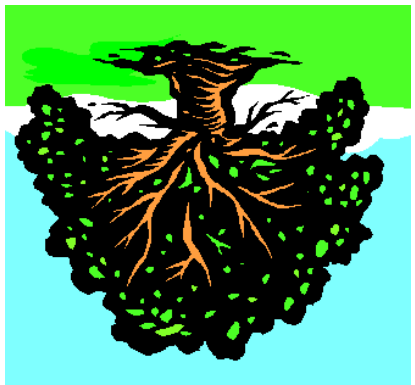
- Modell
  - Ordervägarna i ett regemente eller företag (ordnat träd).
  - Stamtavla/släktträd (binärt träd).
- Tillämpningsexempel inom datavärlden:
  - Filsystem.
  - Klasshierarkier i Java/C++.
  - Besluts-/sök-/spelträd inom AI.
  - Prologs exekvering.
  - Problemlösning:



Träd, terminologi

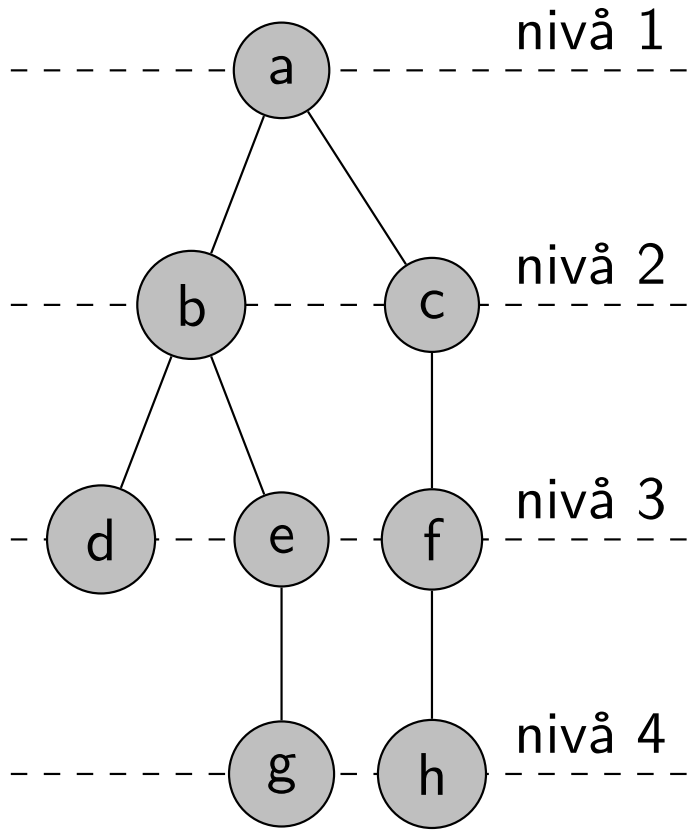
- Varje träd har en rot (*root*).
- Ett träd består av noder (*nodes*).
- Om det finns flera noder så finns det också grenar (*branches*).
- Noder längst ut på grenarna kallas löv (*leaves*).





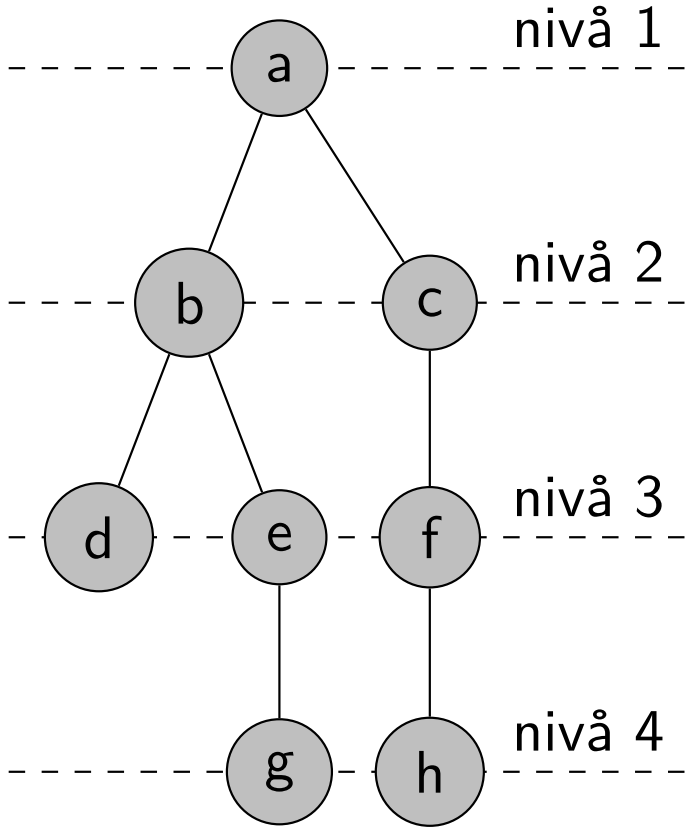
Träd, organisation (1)

- Elementen i ett träd kallas för noder.
- En nod har en position och ev. ett värde.
- Värdet på en nod kallas etikett (*label*).
- Ett träds noder finns på olika nivåer (*levels*).
- Ett träd är organiserat som en föräldra-barn-hierarki:
  - Ett barn ligger på nivån under dess förälder.
  - Alla noder på en nivå med samma förälder kallas syskon (*sibling*).
- Ett delträd = en nod och dess avkomma.



Träd, organisation (2)

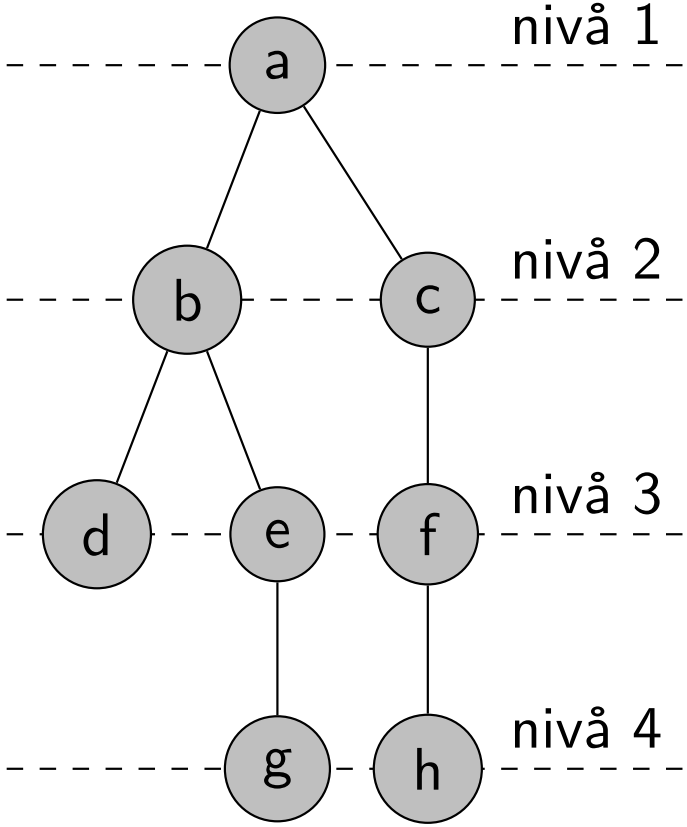
- Höjden $h(x)$ för nod x är antalet bågar på den längsta grenen i det träd där x är rot:
  - "Hur långt är det ner?"
- Djupet $d(x)$ hos en nod x är antalet bågar från x upp till roten:
  - "Hur långt är det upp?"



Träd, höjd

- 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

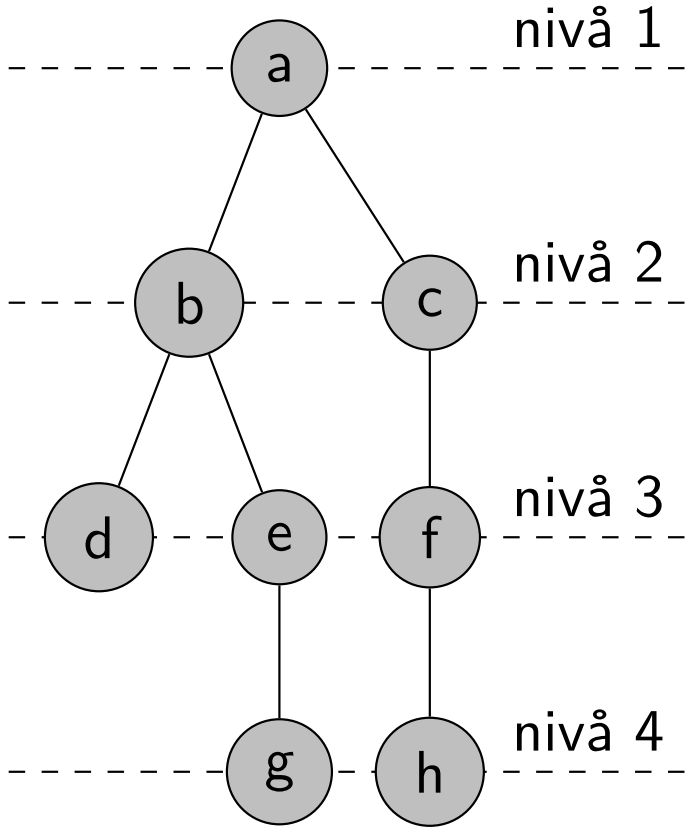
$$h(T) = h(roten)$$
$$h(g) = 0,$$
$$h(b) = 2,$$
$$h(a) = 3 = h(T).$$



Träd, djup

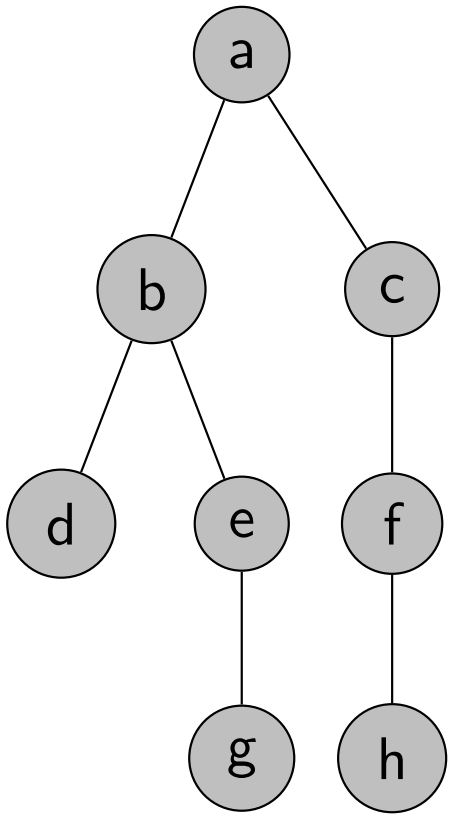
- Djupet $d(x)$ hos en nod x är antalet bågar från x upp till roten:

$$d(a) = 0,$$
$$d(b) = 1,$$
$$d(h) = 3,$$
$$\textit{nivå}(x) = d(x) + 1.$$



Träd, globala egenskaper

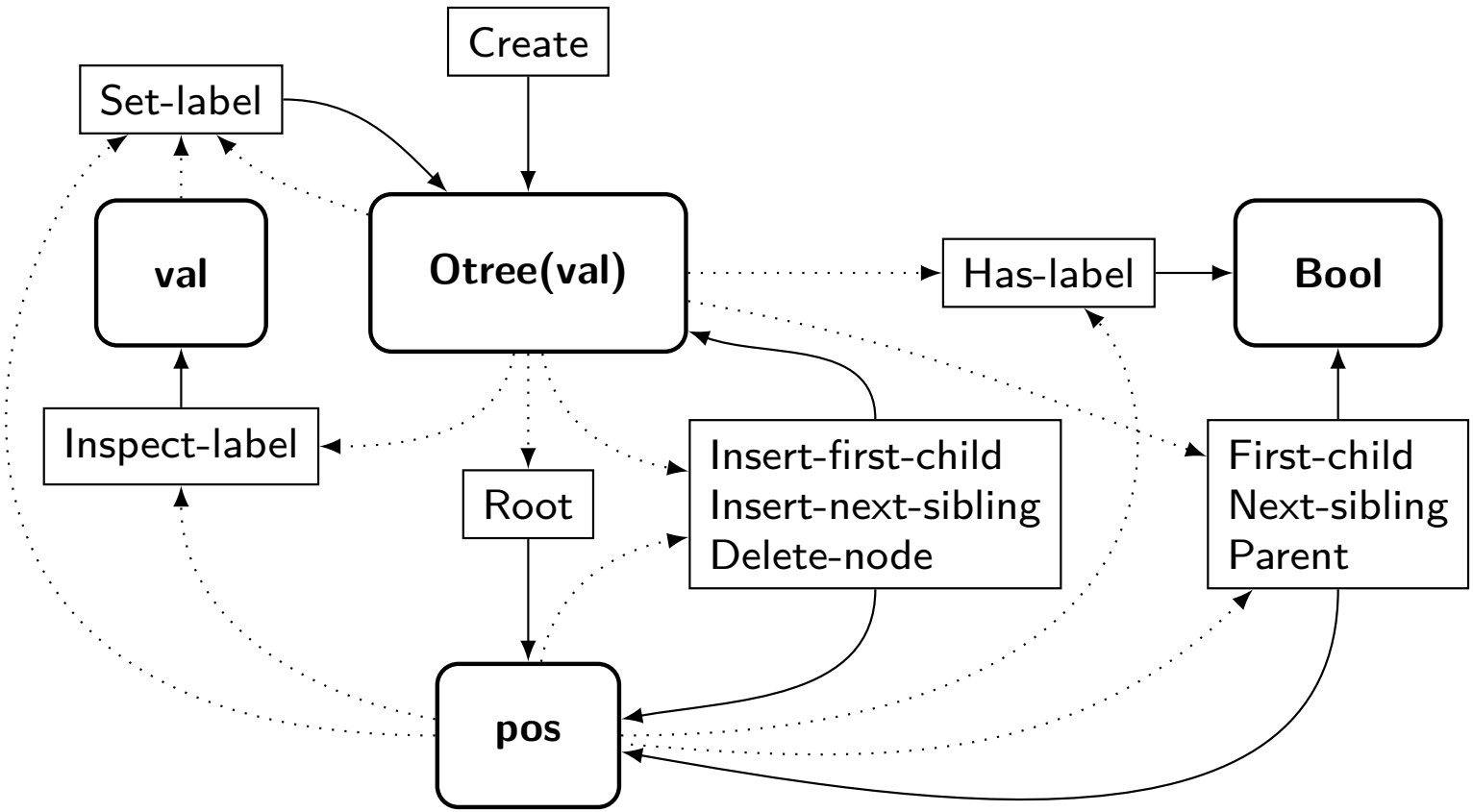
- Ett träd har ett ändligt antal noder
- Ett träd är en homogen datatyp.
- Ett träd saknar cykler, dvs. vägen mellan två noder är alltid unik.
- Ett träd är en rekursiv datatyp; varje delträd är i sig ett träd.



Specifikation av träd

- Navigeringsorienterad
  - Om man arbetar med enstaka träd som förändras långsamt, löv för löv, så är navigeringsorienterad specifikation bättre.
    - Naturligt med operationer som `Insert-node`, `Delete-node`.
- Delträdsorienterad
  - Håller man på med träd och delträd som man vill dela upp eller slå samman är delträdsorienterad bättre.
    - Naturligt med operationer som `Join`, `Split`.
- Vi kommer att fokusera på den navigeringsorienterade specifikationen.

Signaturdiagram för ordnat träd



Olika typer av träd (1)

- Ordnat träd, t.ex. familjeträd:
  - Syskonen är linjärt ordnade.
  - Syskonen kan representeras av en lista.
- Oordnat träd, t.ex. filsystemet på en dator:
  - Ordningen mellan syskonen är odefinierad.
  - Syskonen kan representeras av en mängd.
- Urträd:
  - Mer abstrakt än de två förra. Har en egen datatyp som hanterar syskonen.

Olika typer av träd (2)

- Oriktade träd
  - Kan navigera lika lätt upp och ner i trädet.
- Riktade träd
  - Kan bara gå i en riktning i trädet.
  - I ett nedåtriktat träd saknas `Parent`.
  - I ett uppåtriktat träd saknas `Children`.
  - Ett uppåtriktat träd måste ha en funktion för att nå något annat än roten, t.ex. en operation som returnerar alla löv.
- Binära träd, t.ex. stamtavla
  - Varje nod har högst två barn.

Om ordning

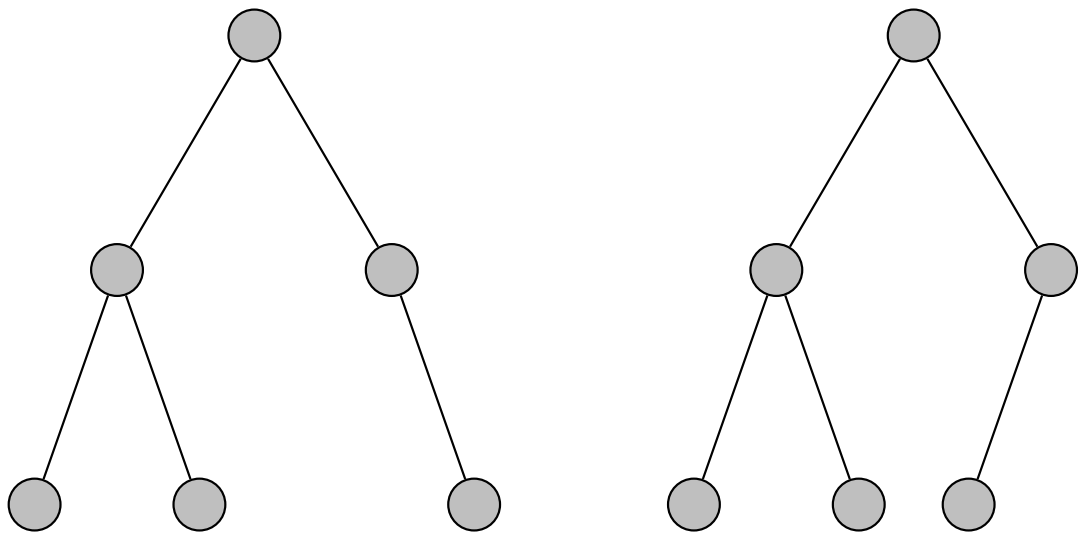
Ordnad Används för att beskriva olika sätt att ordna element före/efter varandra i ett objekt i en datatyp.

Riktad När det finns en asymmetri när det gäller operationer för att hitta från ett element till ett annat.

Sorterad När elementvärdena är ordnade enligt någon ordningsrelation definierad för elementens värden (ex. "äldre än").

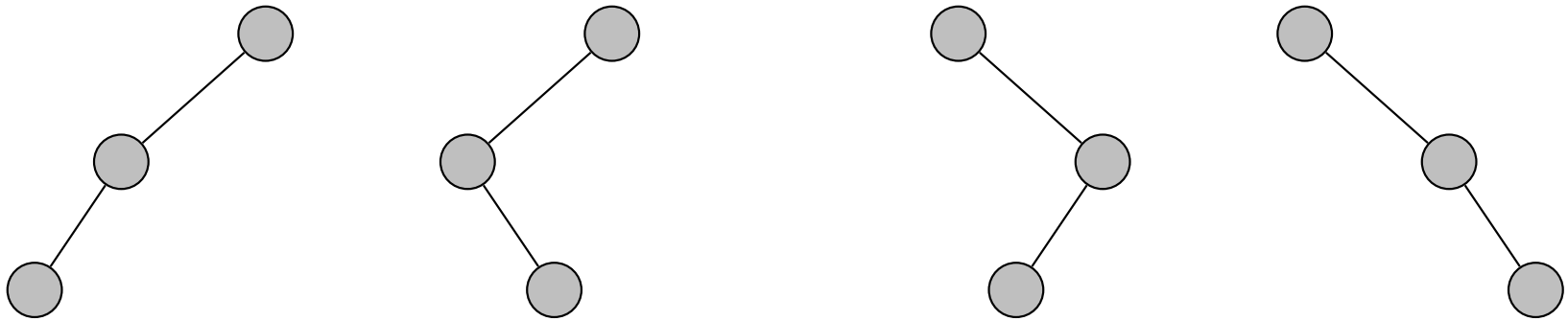
Binära träd

- En nod i ett binärt träd kan ha högst två barn.
  - Barnen kallas vänster- och högerbarn.
  - Ordningen mellan barnen är odefinierad, även om träden oftast presenteras med vänsterbarnet "före" (till vänster) om högerbarnet.
  - Två olika binära träd kan vara samma "ordnade träd med max två barn".



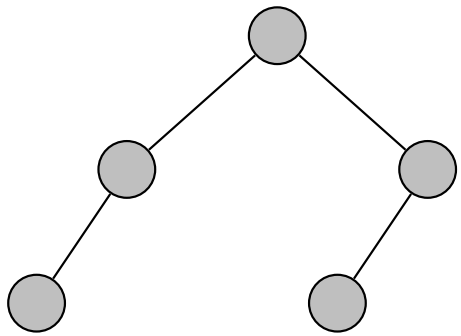
Binära träd, maximal och minimal höjd

- Maximal höjd: $n - 1$, en nod per nivå:

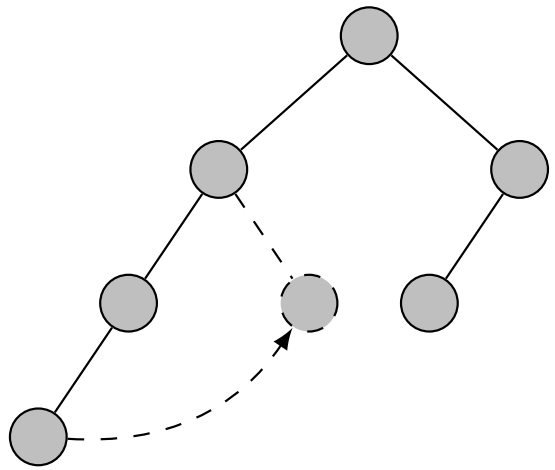


- Minimal höjd: det går ej att flytta några noder och få en mindre höjd:

minimal höjd

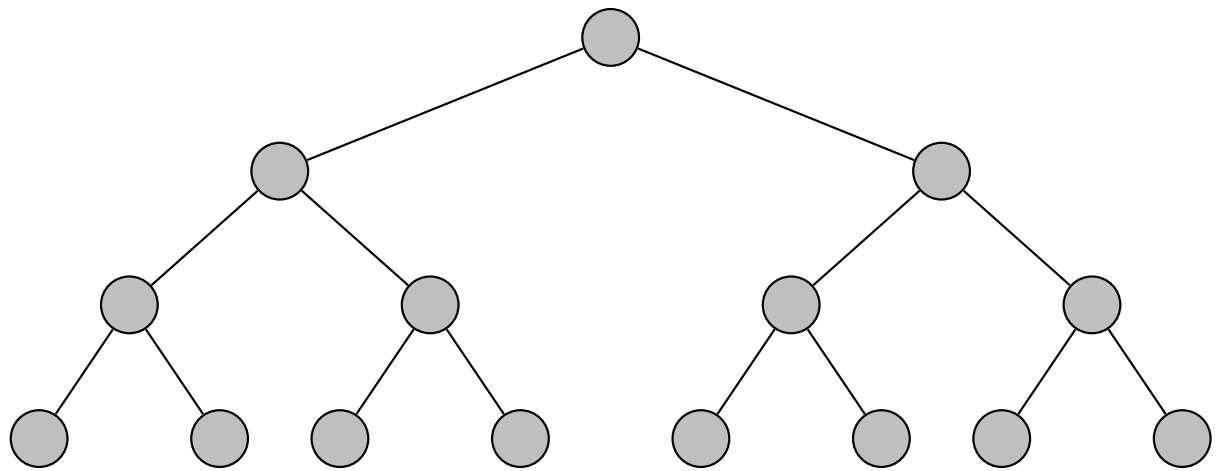


ej minimal höjd



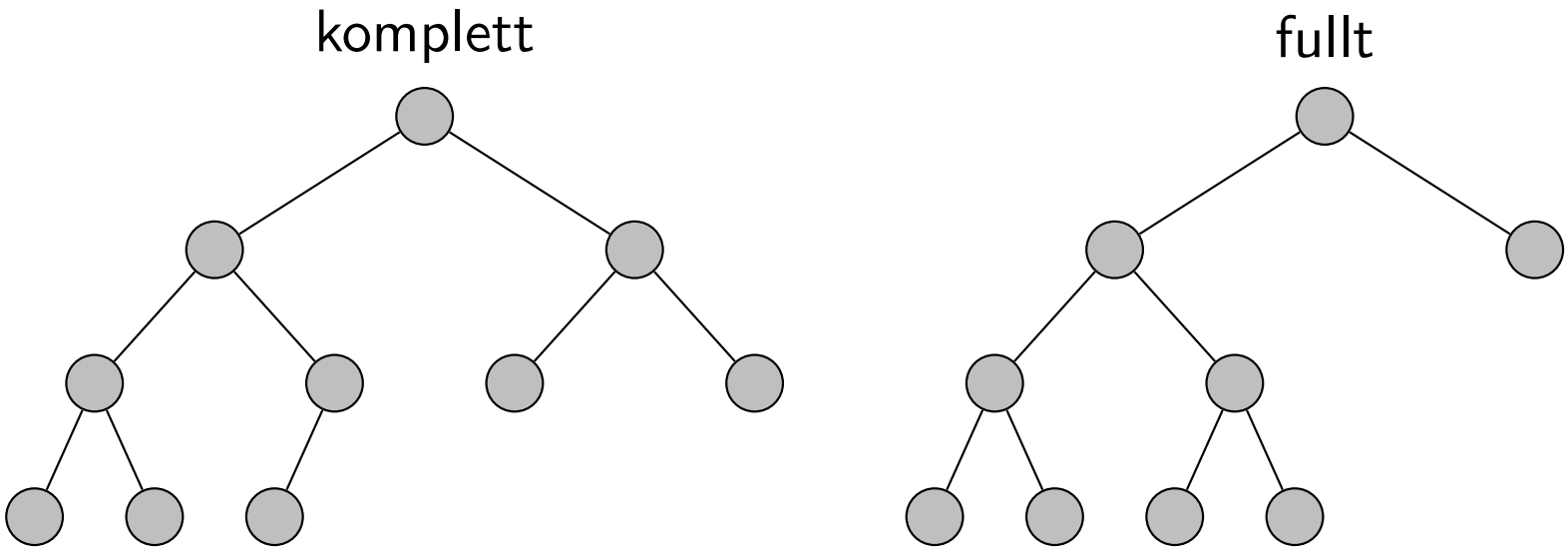
Binära träd, höjd och antal noder

- 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
    - $\log_2(n+1) - 1 \leq h < \log_2(n+1)$,
    - h är alltså av $O(\log_2(n))$.



Binära träd, balanserade träd

- Man vill ofta ha så grunda träd som möjligt:
  - Om vänster och höger delträd har ungefär lika många noder har trädet balans.
  - I ett balanserat träd är vägen till en slumpvis vald nod $O(\log_2 n)$.
- 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.



Trädalgoritmer

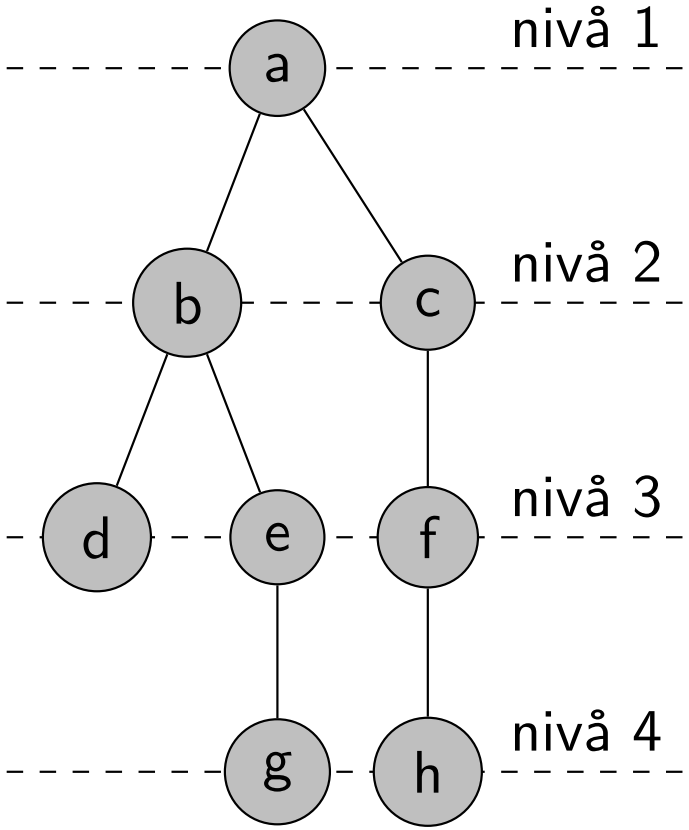
- Basalgoritmer:
  - Beräkna djup.
  - Beräkna höjd.
  - Slå ihop två träd.
  - Dela upp ett träd.
  - Traversera (förflytta sig i) trädet.
  - Beräkna/evaluera etikett(-er) i trädet.

Traversering av träd

- Tillämpningar av träd involverar ofta att man
  - söker efter ett element med vissa egenskaper,
  - filtrerar ut element med vissa egenskaper, eller
  - transformerar strukturen till en annan struktur
    - Exempelvis sortering och balansering.
- Alla dessa bygger på att man traverserar strukturen.
- Det finns två grundläggande traverseringsmetoder: bredden-först och djupet-först.

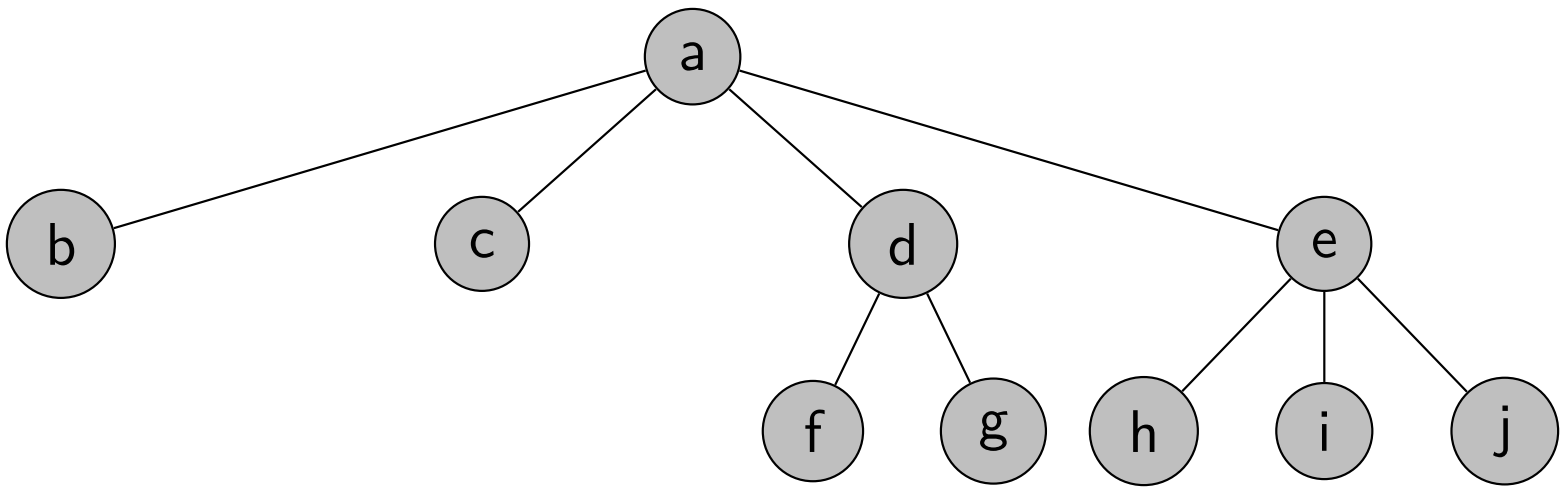
Traversering av träd, bredden-först

- Trädet undersöks en nivå i taget.
- Först roten, sedan rotens barn, dess barnbarn, etc.
- En kö är ofta hjälp i implementationen.
- Varje nod i trädet besöks endast en gång, dvs. $O(n)$.



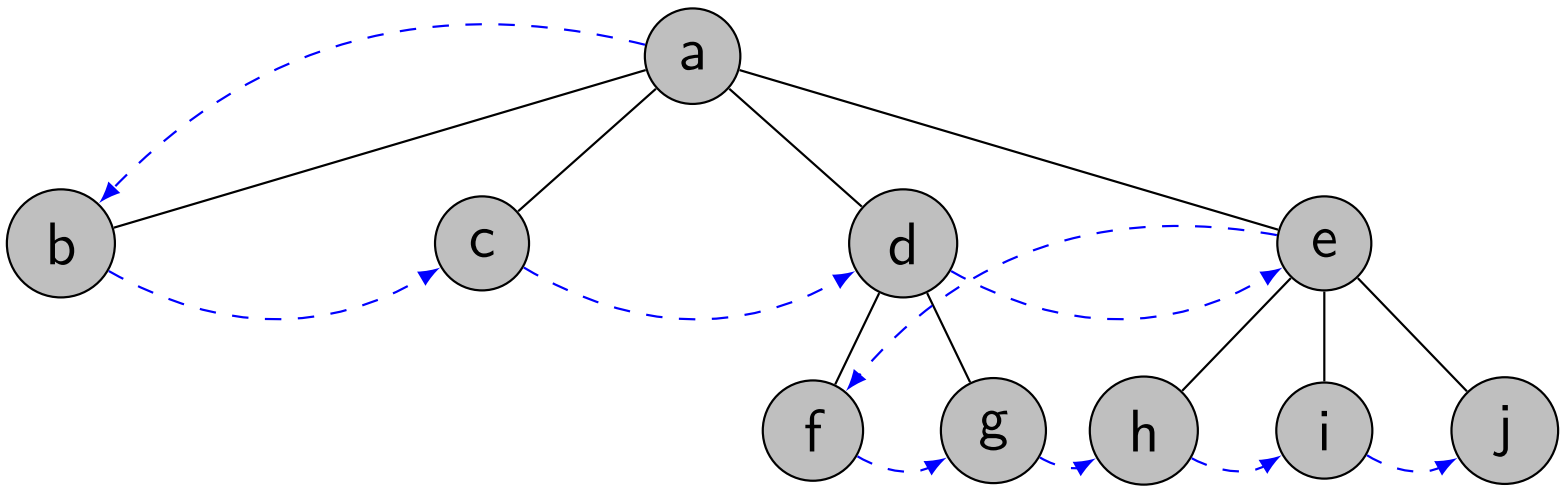
Traversering av träd, bredden-först, exempel

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



Traversering av träd, bredden-först, exempel

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

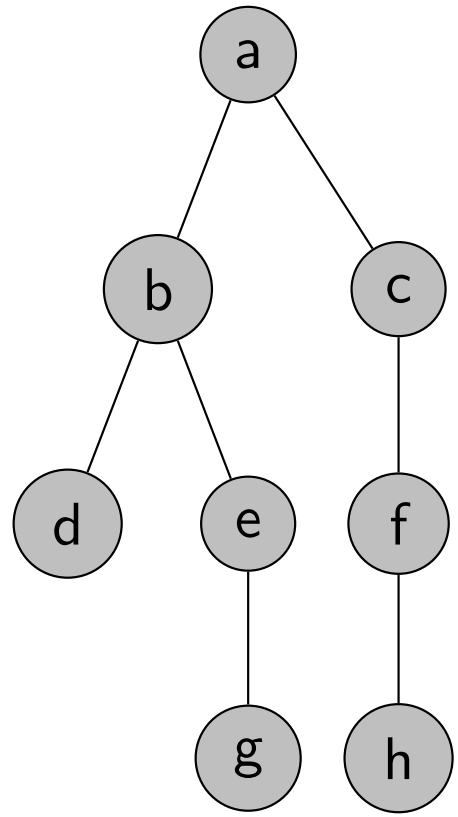


Ordning: a, b, c, d, e, f, g, h, i, j.

Traversering av träd, djupet-först

- Man följer varje gren i trädet från roten till lövet.
- En stack är ofta till hjälp vid implementationen.
- Varje nod i trädet besöks endast en gång, dvs. $O(n)$.
- Tre varianter på traversering:

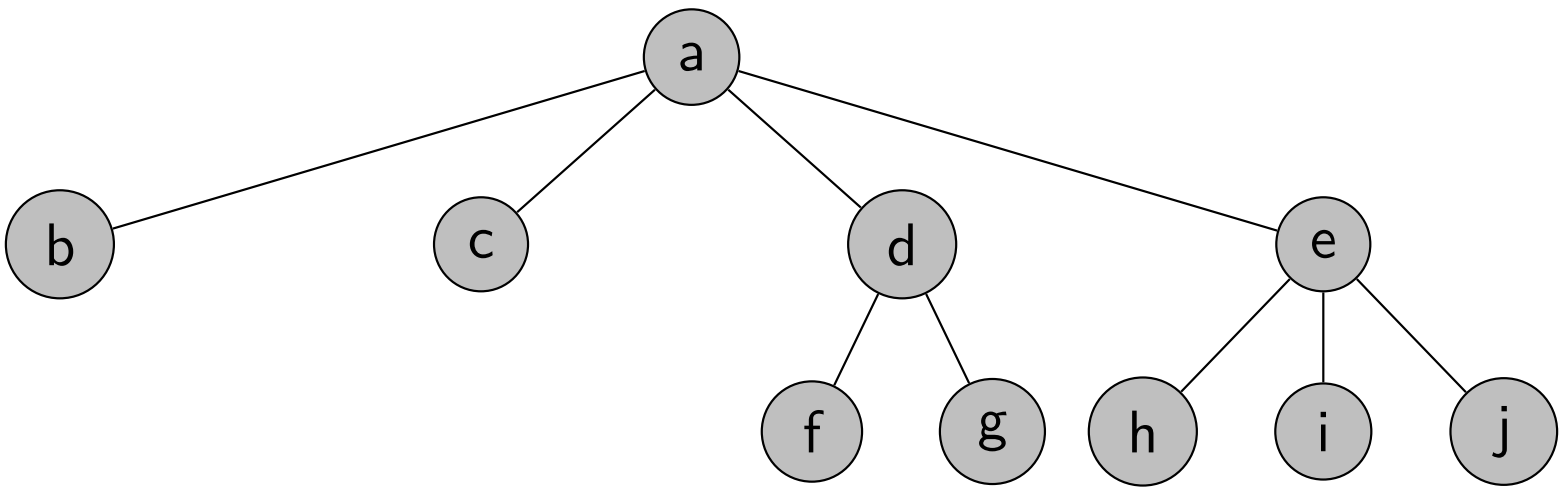
Preorder	label, child 1, child 2, . . . , child n_i
Postorder	child 1, child 2, . . . , child n_i, label
Inorder	child 1, label, child 2, . . . , child n_i



Traversering av träd, djupet-först, preorder

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

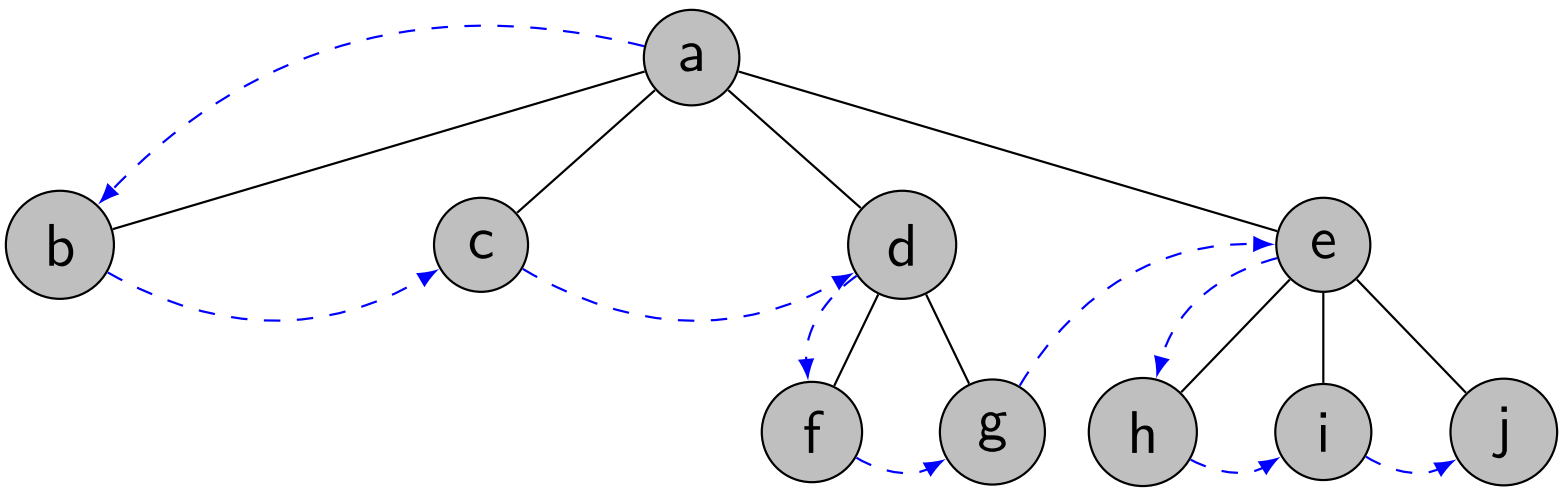
```
preOrder(BinTree T)
    input: A binary tree T to be trave
compute(root(T))
preOrder(leftChild(T))
preOrder(rightChild(T))
```



Traversering av träd, djupet-först, preorder

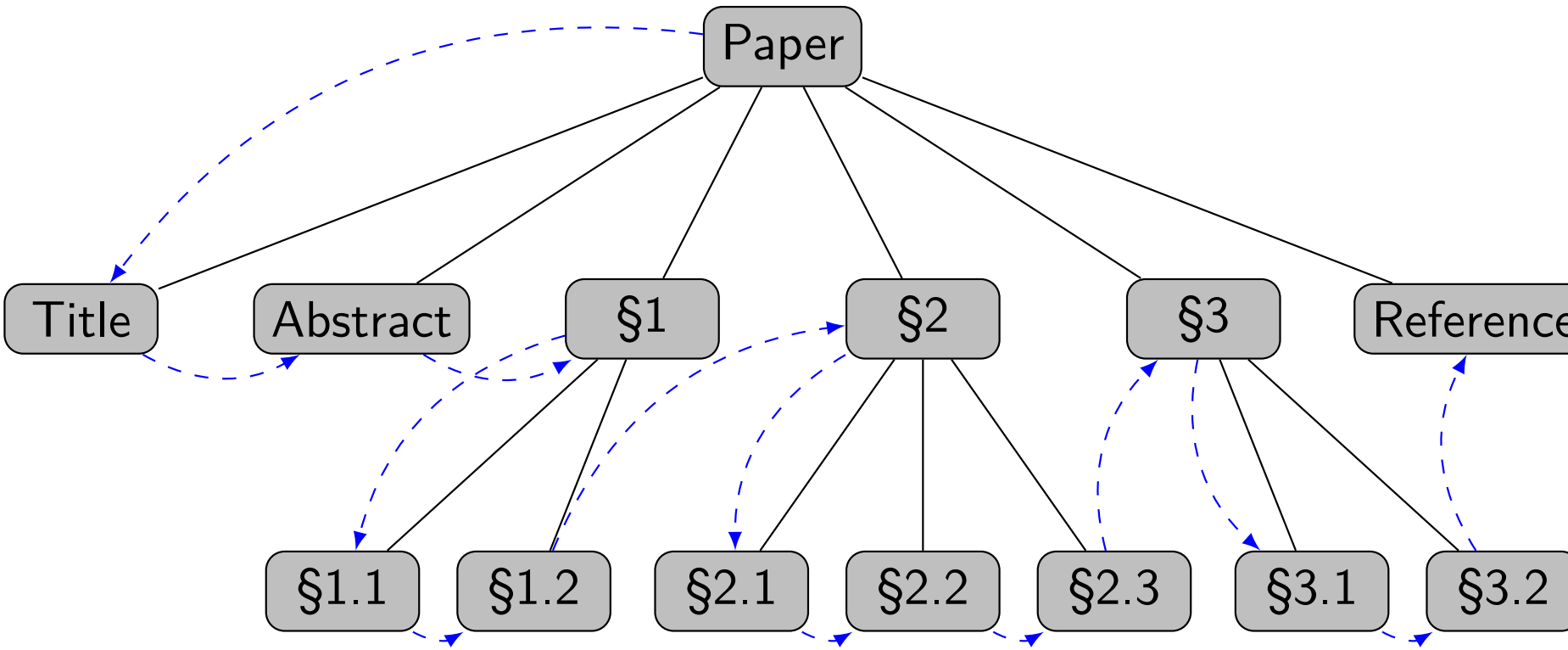
```
Algorithm preOrder(Tree T)
    input: A tree T to be traversed
compute(root(T))// Do something with
                // the root node.
for each child c of root(T) do
    preOrder(c)
```

```
preOrder(BinTree T)
    input: A binary tree T to be trave
compute(root(T))
preOrder(leftChild(T))
preOrder(rightChild(T))
```



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

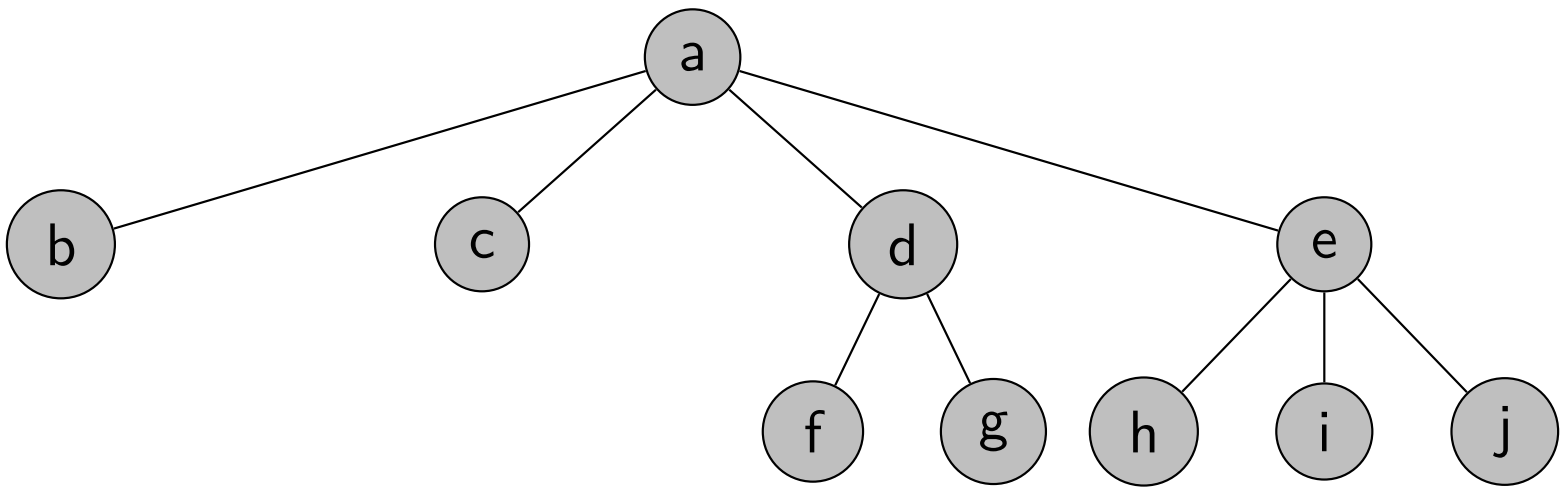
Preorder — läsa ett dokument



Traversering av träd, djupet-först, postorder

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

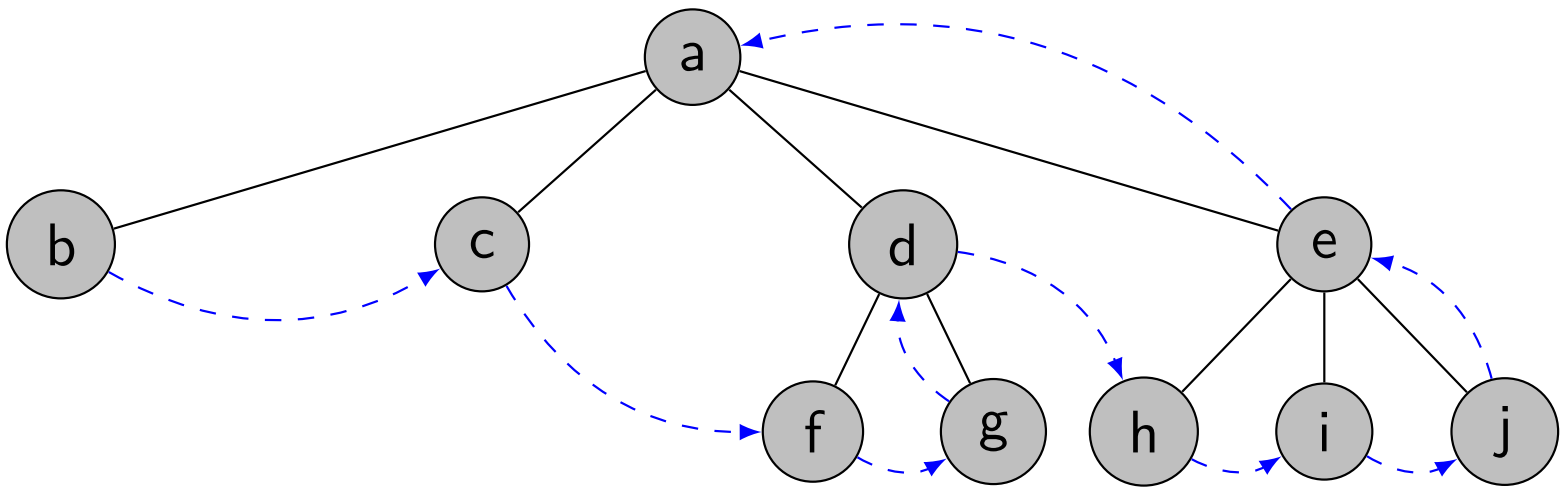
```
postOrder(BinTree T)
    postOrder(leftChild(T))
    postOrder(rightChild(T))
    compute(root(T))
```



Traversering av träd, djupet-först, postorder

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

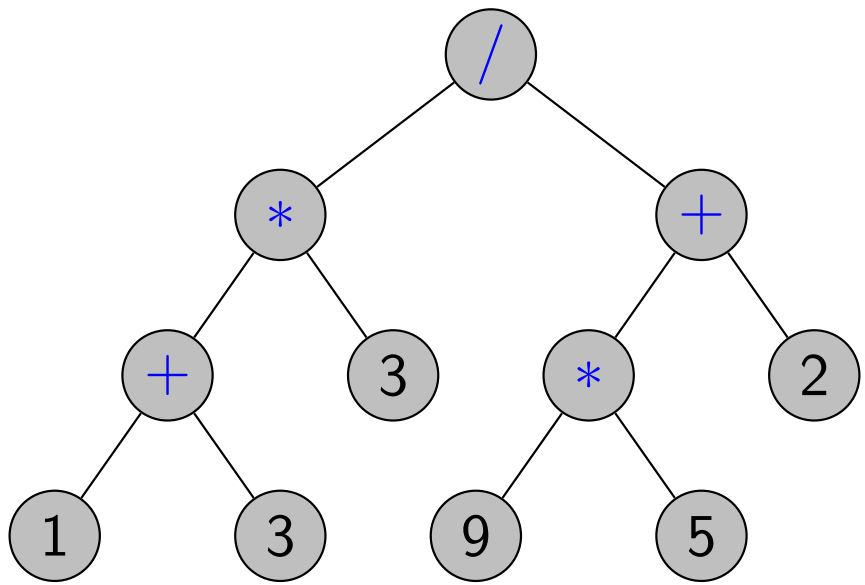
```
postOrder(BinTree T)
    postOrder(leftChild(T))
    postOrder(rightChild(T))
    compute(root(T))
```



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

Postorder — Beräkna aritmetiska uttryck utan paranteser

```
Algorithm evaluateExpression(BinTree T)
If isLeaf(T)
    return getValue(T)
else
    x ← evaluateExpression(leftChild(T))
    y ← evaluateExpression(rightChild(T))
    op ← getValue(T)
    return x op y
```

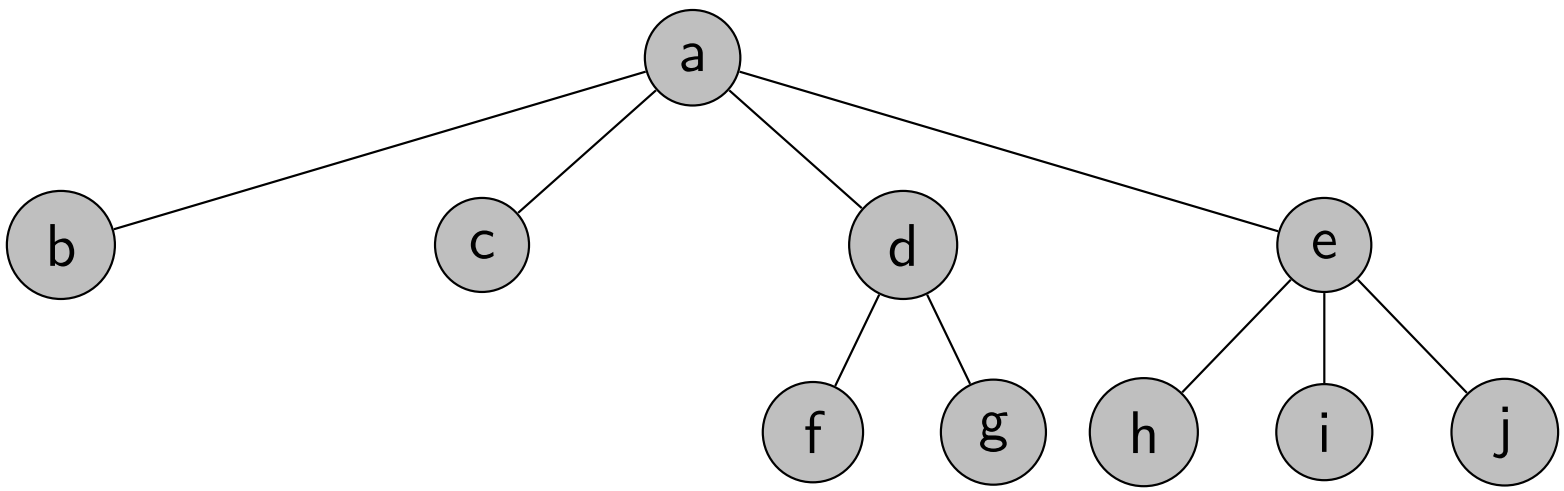


Order: 1, 3, +, 3, *, 9, 5, *, 2, +, /.

Traversering av träd, djupet-först, inorder

```
Algorithm inOrder(Tree T)
    input: A tree T to be traversed
inOrder(firstChild(T))
compute(root(T))// Do something with
                // the root node.
for each child c (- first) of root(T) do
    inOrder(c)
```

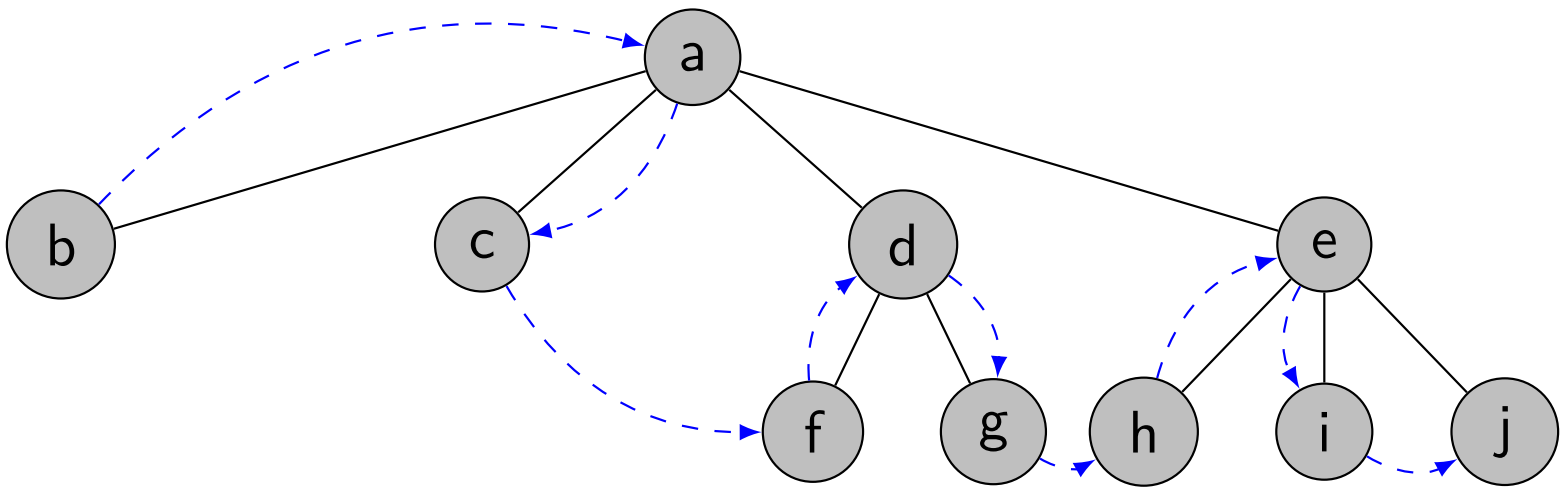
```
inOrder(BinTree T)
inOrder(leftChild(T))
compute(root(T))
inOrder(rightChild(T))
```



Traversering av träd, djupet-först, inorder

```
Algorithm inOrder(Tree T)
    input: A tree T to be traversed
inOrder(firstChild(T))
compute(root(T))// Do something with
                // the root node.
for each child c (- first) of root(T) do
    inOrder(c)
```

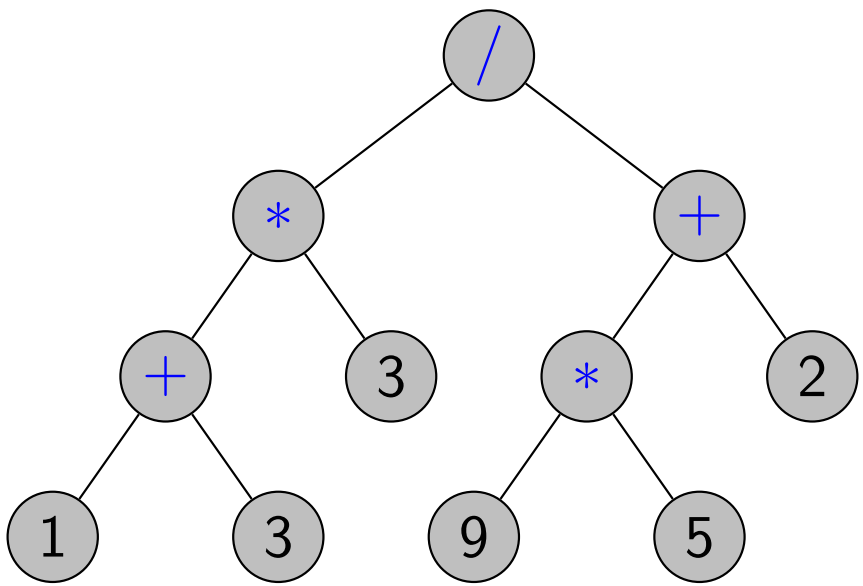
```
inOrder(BinTree T)
inOrder(leftChild(T))
compute(root(T))
inOrder(rightChild(T))
```



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

Inorder — Skriva ut aritmetiska uttryck

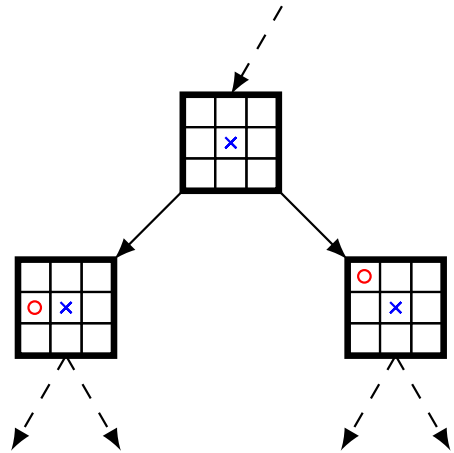
```
Algorithm printExpression(Tree T)
print "("
If hasLeftChild(T)
    printExpression(leftChild(T))
print getValue(T)
If hasRightChild(T)
    printExpression(rightChild(T))
print ")"
```



Utskrift: ((1+3)*3)/((9*5)+2)

Träd, tillämpningar

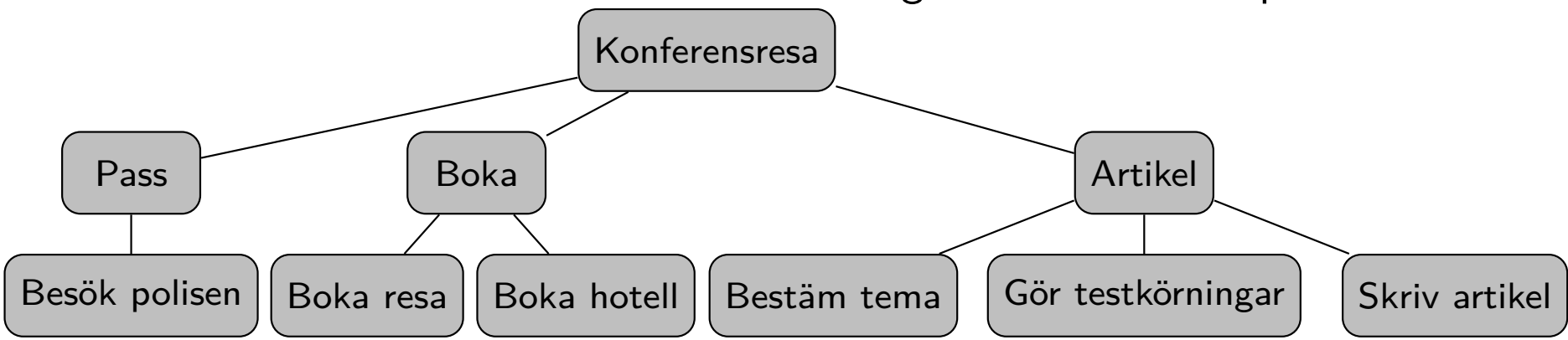
- Konstruktioner av andra typer (speciellt binära träd).
- Sökträd:
  - Varje nod symboliserar ett givet tillstånd.
  - Barnen symboliserar de olika tillstånd man kan hamna i utifrån förälderns tillstånd.



  - 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 ev. i vilken ordning man bör lösa delproblem.

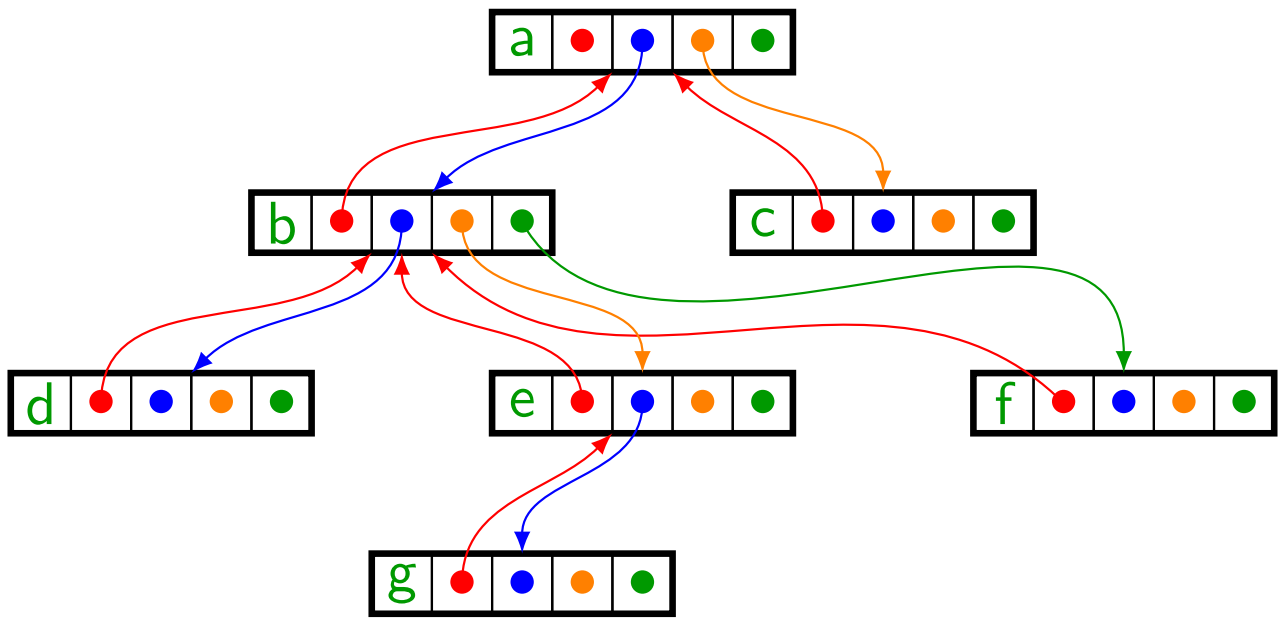


  - Ofta använder man OCH/ELLER-träd där man kan ha OCH-kanter eller ELLER-kanter mellan förälder och barn:

    OCH alla barn behövs för lösningen.
    ELLER något barn behövs för lösningen.

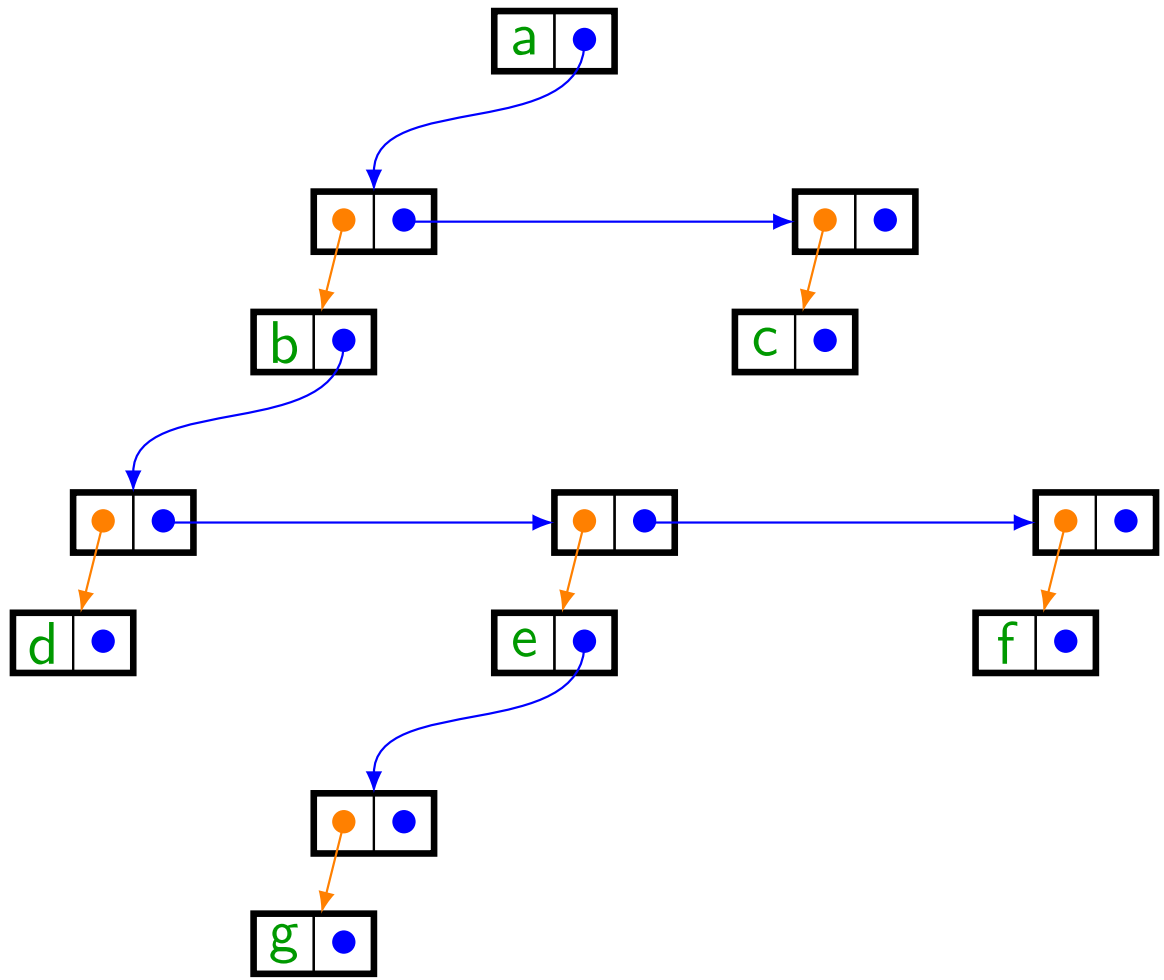
Konstruktioner av träd (1)

- Ordnat träd som k-länkad struktur:
  - Noden i trädet består av k-celler med etikett, {länk till föräldern} och $k-1$ länkar till barnen.
    - Antalet noder i trädet dynamiskt.
    - Maximala antalet barn $k-1$ bestämt i förväg.
    - Om det är stor variation i antalet barn så finns outnyttjade länkar.
- Exempel för $k = 4$, 4-cell (etikett, förälder, första barn, andra barn, tredje barn).



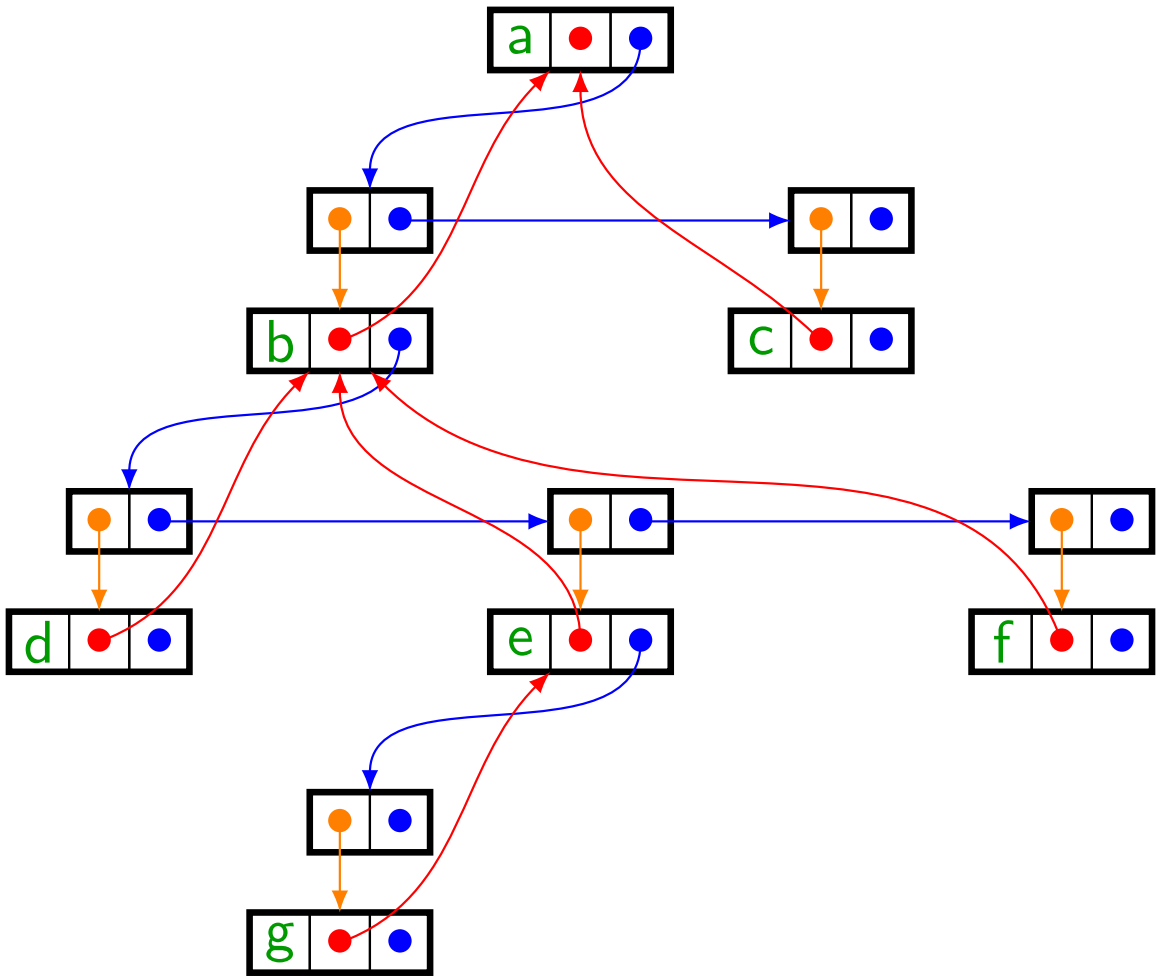
Konstruktioner av träd (2)

- Nedåtriktat ordnat träd som 1-länkad struktur med lista av barn:
  - Noden får en etikett och länk till lista av barn.
    - Antalet noder i trädet dynamiskt.
    - Antalet barn dynamiskt.



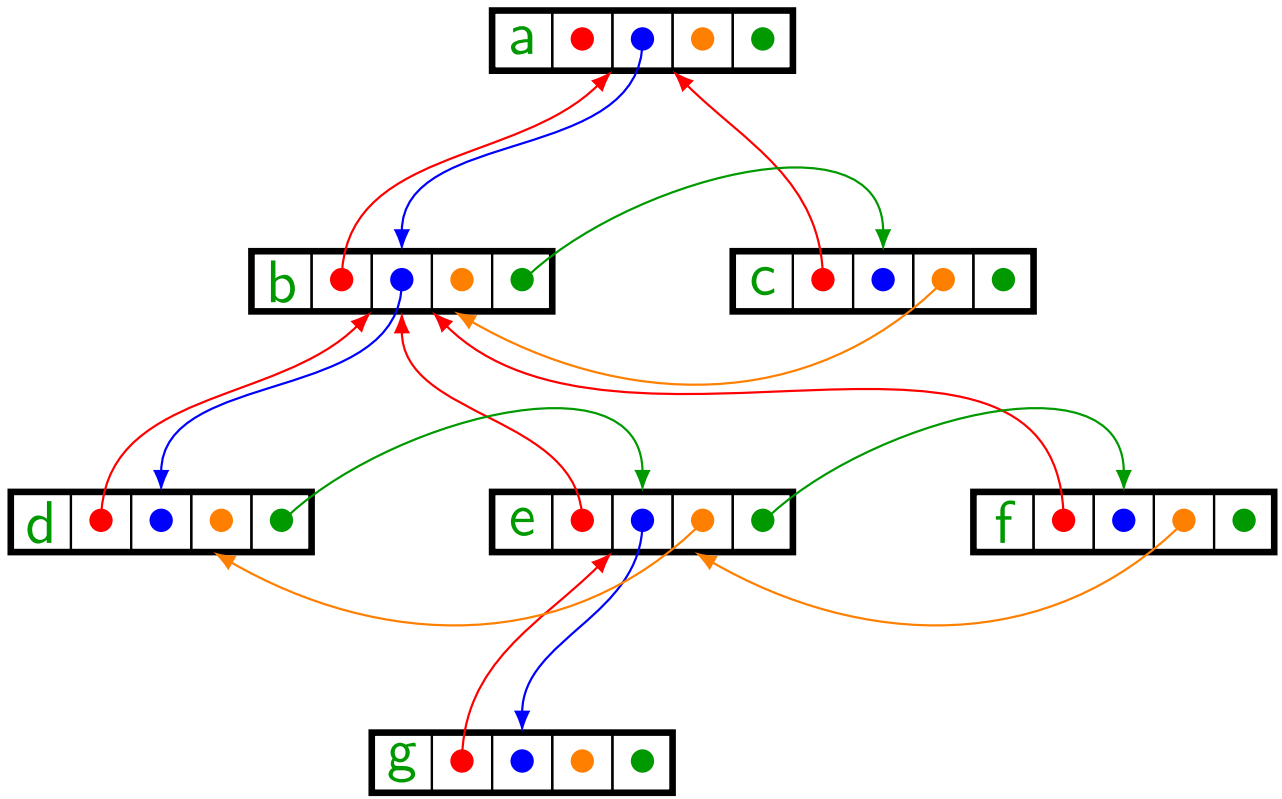
Konstruktioner av träd (3)

- Utöka till 2-celler så blir trädet oriktat:
  - Noden får en etikett, länk till föräldern, samt länk till lista av barn.



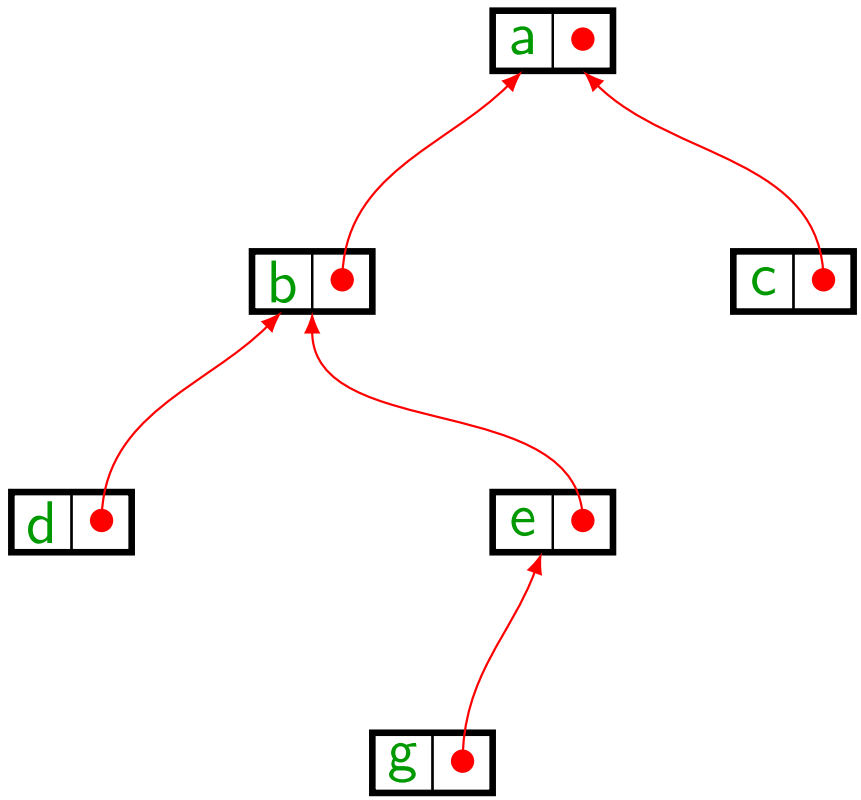
Konstruktioner av träd (4)

- Oriktat träd med hjälp av 4-cell (etikett, förälder, första barn, föregående syskon, efterföljande syskon).



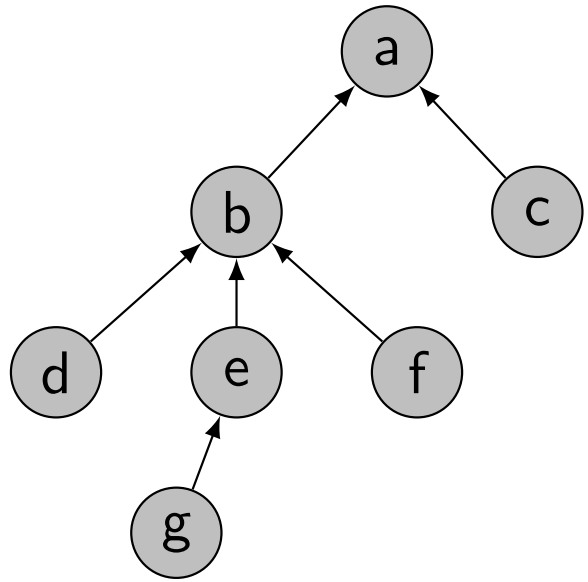
Konstruktioner av träd (5)

- Uppåtriktat träd med hjälp av 1-cell (etikett, förälder).



Konstruktioner av träd (6)

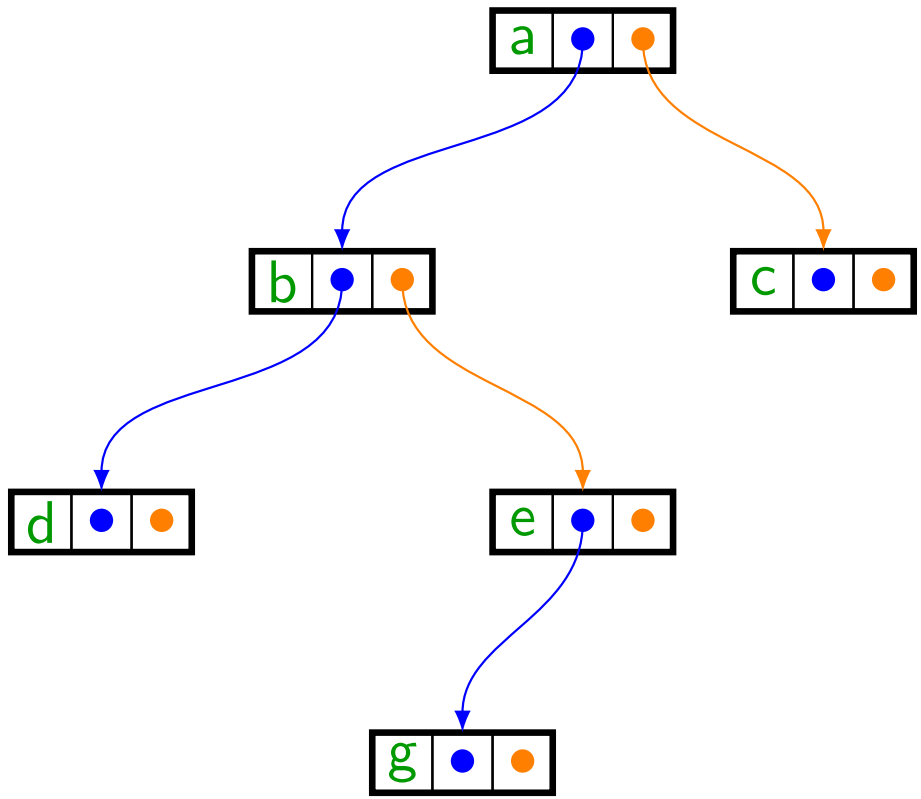
- Oordnat uppåtriktat träd som fält:
  - Varje element i en vektor består av ett par: nodens etikett och en referens till föräldern.
    - Tar liten plats.
    - Inget bra stöd för traversering (t. ex. svårt avgöra vilka noder som är löv).
    - Maximala storleken på trädet måste bestämmas i förväg.



	Etikett	Förälder
1	c	4
2	e	8
3		-1
4	a	0
5	g	2
6		-1
7	d	8
8	b	4
9	f	8

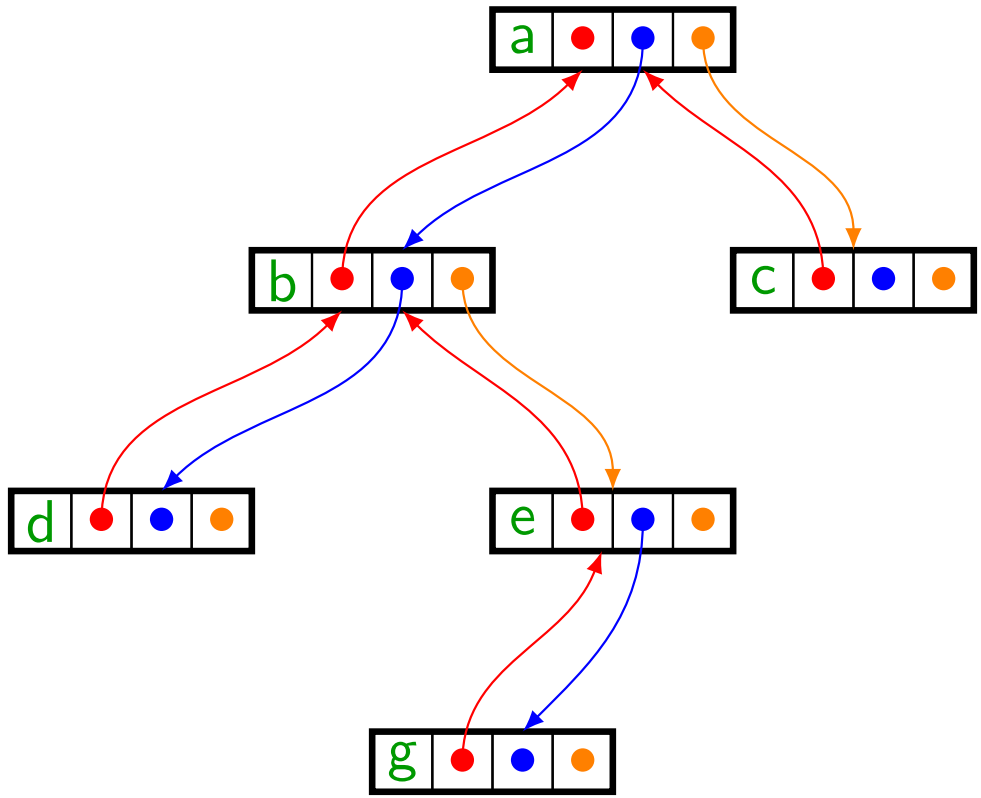
Konstruktioner av binära träd (1)

- Nedåtriktat binärt träd med hjälp av 2-cell (etikett, vänsterbarn, högerbarn).

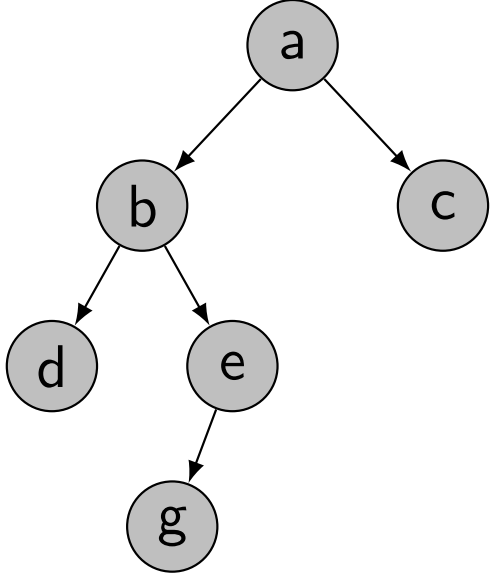


Konstruktioner av binära träd (2)

- Oriktat binärt träd med hjälp av 3-cell (etikett, förälder, vänsterbarn, högerbarn).



Konstruktioner av binära träd (3)



- Binärt träd som fält
  - Roten har index 1 och noden med index i har
    - sitt vänsterbarn i noden med index $2i$,
    - sitt högerbarn i noden med index $2i + 1$,
    - sin förälder i noden med index $\lfloor \frac{i}{2} \rfloor$.
- Tar inget utrymme för strukturinformation.
- Trädet har ett maxdjup (statiskt fält).
- Krävs "markörer" för null och tom nod.
- Ev. slöseri med utrymme.

	Etikett
1	a
2	b
3	c
4	d
5	e
6	-
7	-
8	-
9	-
10	g
11	-
12	-
13	-
14	-
15	-