Trær

I. EKSEMPLER, DEFINISJON

II. BINÆRE TRÆR

III. ADT TRE

IV. BASIS TREALGORITMER (TRAVERSERING)

V. IMPLEMENTASJON AV BINÆRE TRÆR

Lenket Struktur

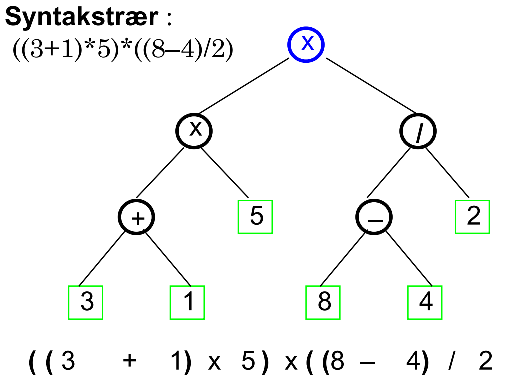
Sequence (Array)

FRA KAP. 6 I LÆREBOKA

Noen eksempler

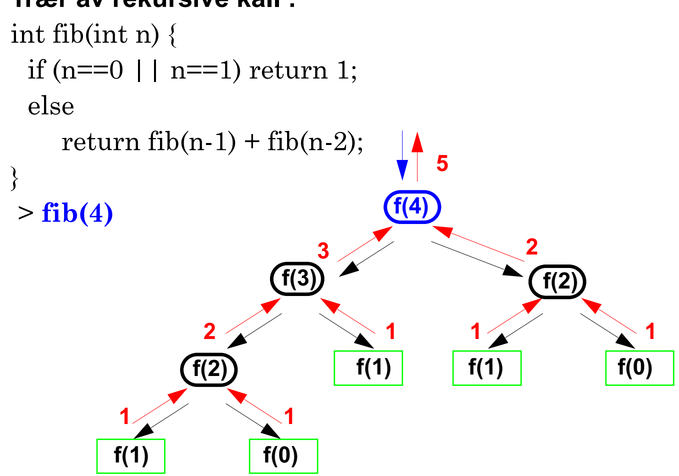
Syntakstrær :

((3+1)*5)*((8–4)/2)



Trær av rekursive kall :

```
int fib(int n) {
  if (n==0 | | n==1) return 1;
  else
     return fib(n-1) + fib(n-2);
}
> fib(4)
```



Filhierarki :

\> **cd**

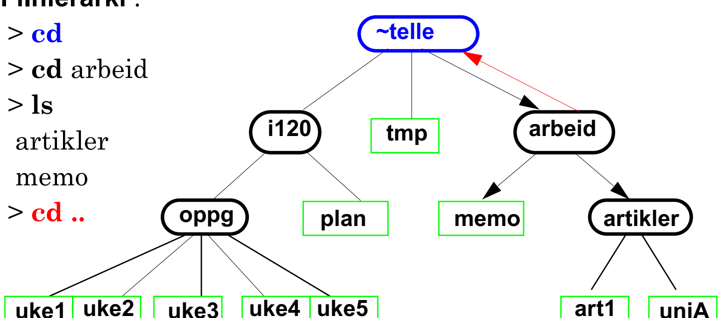
\> **cd** arbeid

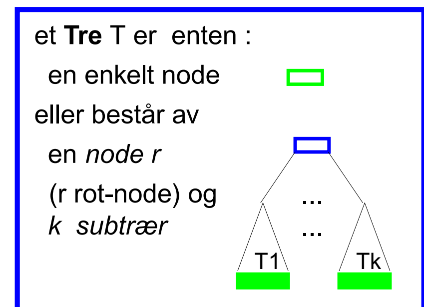
\> **ls**

artikler

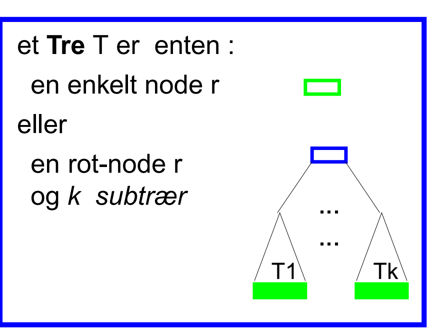
memo

\> **cd ..**





Terminologi (fra slektskapstrær)

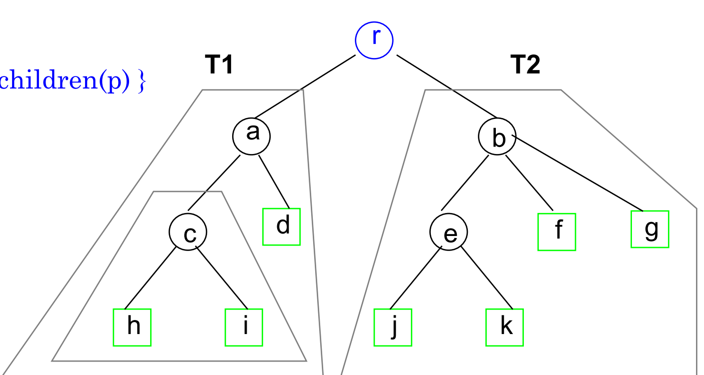


T i figuren har 12 noder (generellt ADT position)**, r** er **roten** i **T**

1. **b** er **foreldrenode** (parent) til **e**, **f**, **g** , og **forgjenger** til **e**, **f**, **g**, **j**, **k**
2. **e**, **f**, **g** er **barna** til **b** : umiddelbare etterfølgere (og disse er **søsken**)
3. **d**, **f**, **g**, **h**, **i**, **j**, **k** **eksterne** noder (**løv**) : har ingen barn
4. **r**, **a**, **b**, **c**, **e** er **interne** noder : ikke løv
5. **dybden** av **e** = 2 : lengden av stien fra roten (**nivå**)
   depth(p) = if isRoot(p) return 0
   else return 1+ depth(parent(p))
6. **høyden** av **b** = 2 : avstand til fjerneste løv under b
   height(p) = if isExternal(p) return 0
   else return 1+ max{ height(x) : x er et av children(p) }
   høyden av **T** = høyden av roten til T
7. **grad** av **b** = 3 : antall barn
8. **T1**, **T2** er **deltrær** av **T**

Noen egenskaper

9. antall kanter = antall noder – 1
10. hver node har en *entydig sti* til roten



Binære Trær

-hver node har **2** eller **0 barn**
-barna er ordnet: **venstre og høyre**

et **Binært Tre** er enten :
en *node* r
eller
en *node* r
og
***2** binære subtrær*
(venstre og høyre)
T1
T2

av og til : **høyst 2** binære subtrær

Noen egenskaper (h: høyde; n: antall noder = # noder)

A. # eksterne noder = # interne noder +1

B. # noder på nivå i $\leq 2^i$

C. h+1 $\leq$ (# eksterne noder) $\leq 2^h$
   $\log_2$ (# eksterne noder) $\leq$ h

D. h $\leq$ (# interne noder) $\leq 2^h - 1$
   $\log_2$ (# interne noder) $\leq$ h

E. $2h+1 \leq n \leq 2^{(h+1)} - 1$
   $\log_2(n+1) - 1 \leq h \leq (n-1)/2$

T

Nivå

0 1 2 3 4

T1 T2

4 2 3 1 0 2 0 0 0 1 0 0 0

package jdsl.core.api;

```
public interface Container
{ Enumeration elements();
  boolean isEmpty(
  int size();
  Container newCo
}
```

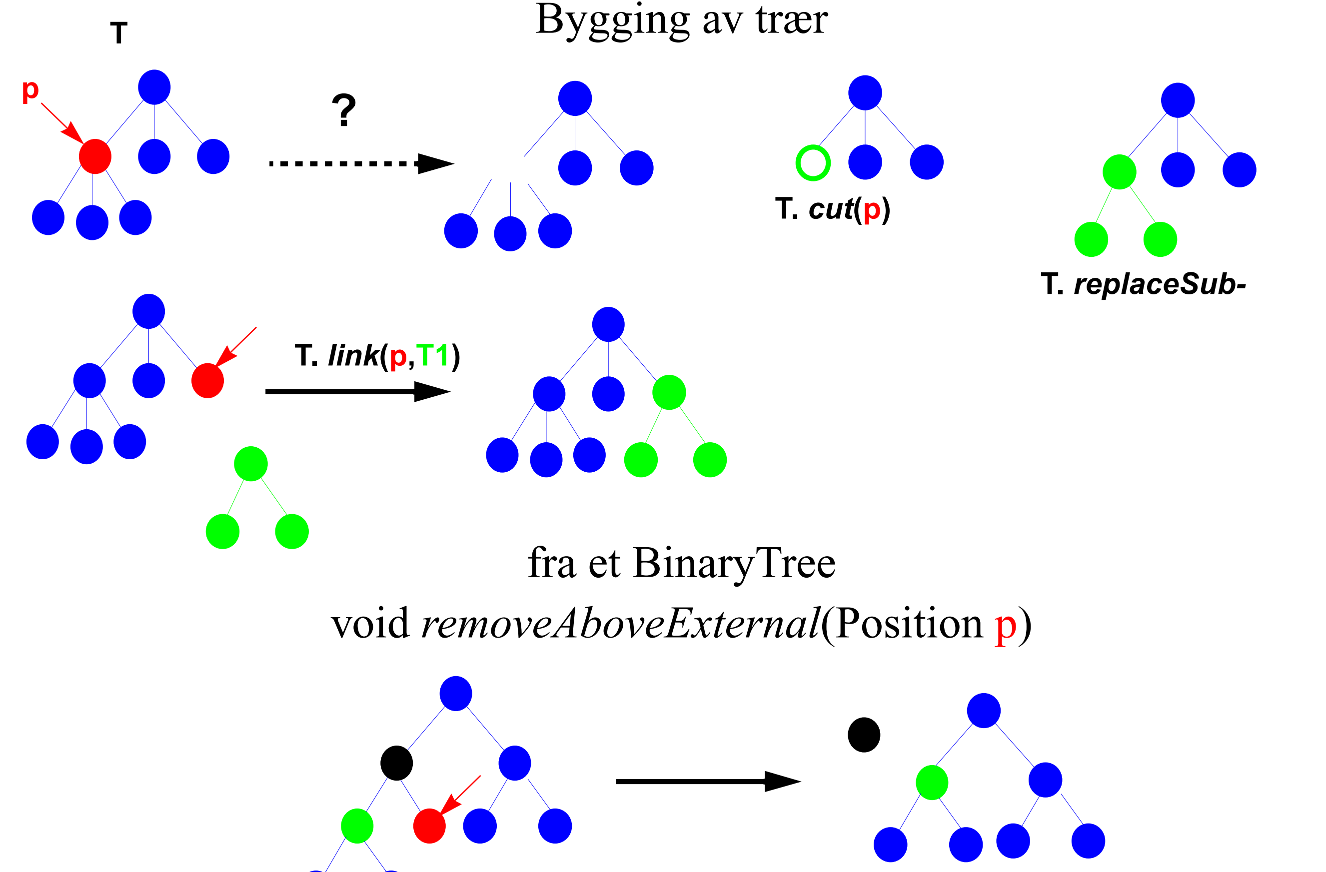
```
public interface PositionalContainer extends Container
{ Enumeration positions();
  void swap(Position p, Position q);
  Object replace(Position p, Object e);
}
```

```
/** Et tre har minst en ekstern node – rot */
public interface InspectableTree
        extends PositionalContainer {
    Position root();
    Position parent(Position v);
/** Barn er ordnet fra venstre til høyre */
    Enumeration children(Position p);
    boolean isInternal(Position v);
    boolean isExternal(Position v);
    boolean isRoot(Position v);
    Enumeration siblings(Position p);
}
```

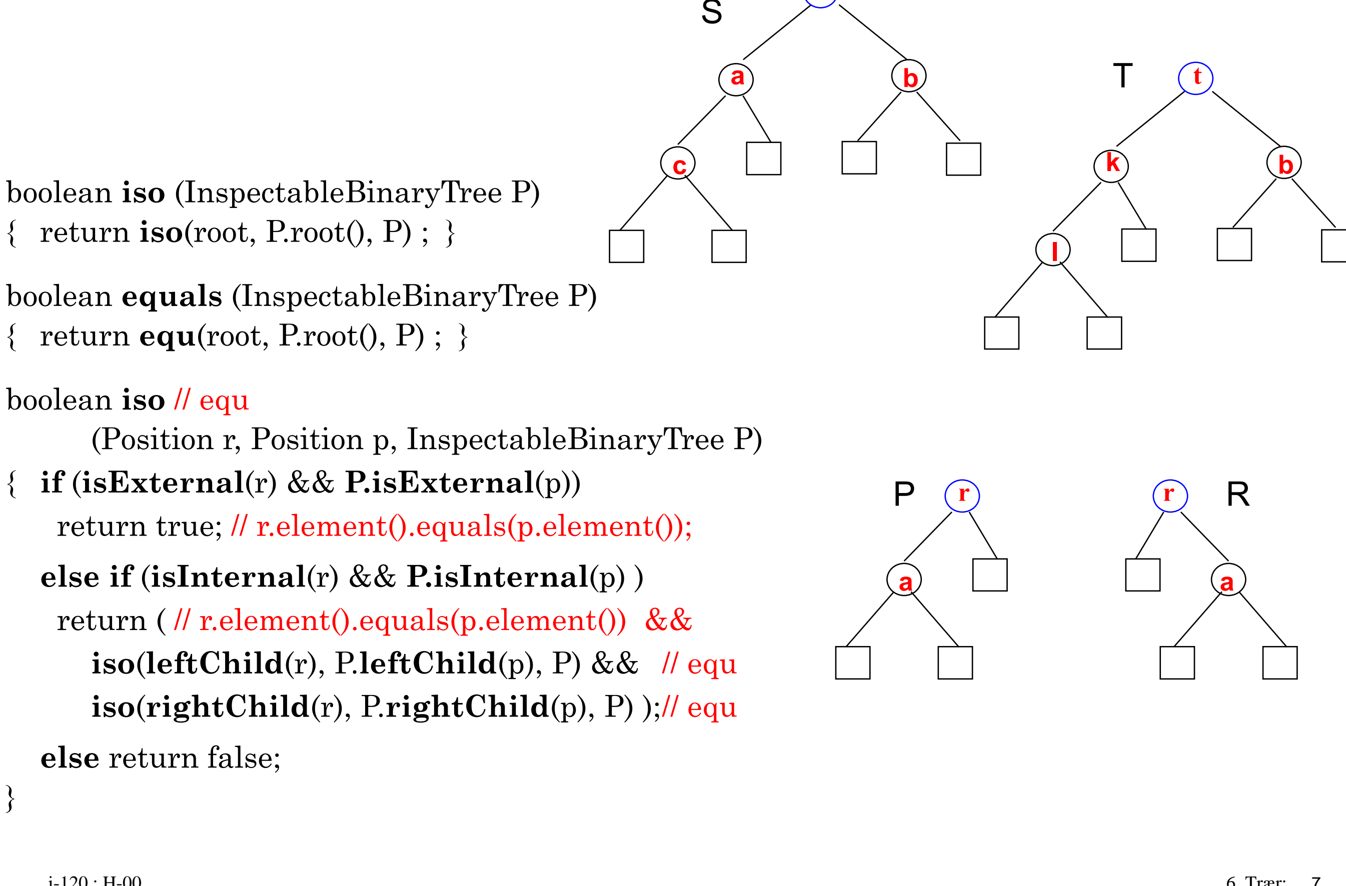
```
public interface InspectableBinaryTree
        extends InspectableTree {
    Position leftChild(Position);
    Position rightChild(Position);
    Position sibling(Position); }
```

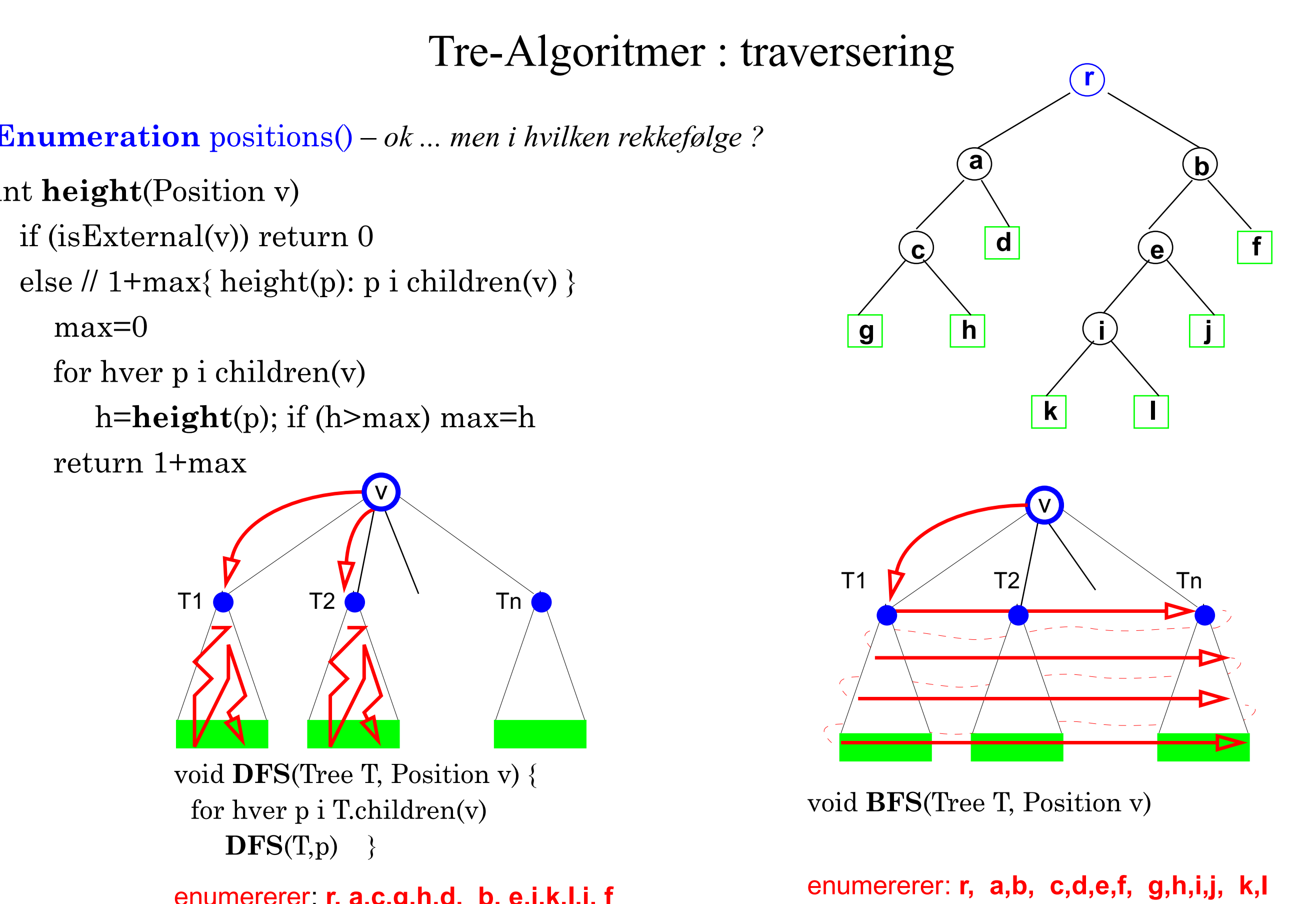
```
public interface BinaryTree
      extends InspectableBinaryTree {
  Tree cut(Position) ;
  void link(Position, BinaryTree) ;
  Tree replaceSubtree(Position, BinaryTree);
  void expandExternal(Position) ;
  void removeAboveExternal(Position) ; }
```

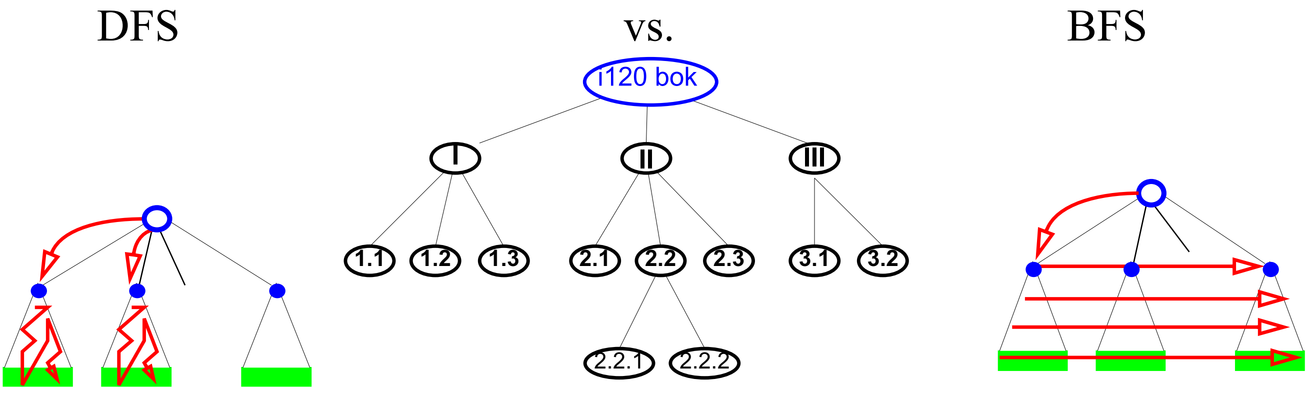
```
public interface Tree extends InspectableTree {
/** Node p erstattes med en ny ekstern node
 *  med null element. */
    Tree cut(Position p);
/** Node p (må være ekstern) erstattes med treet t */
    void link(Positionp p, Tree t);
/** Erstatt subtre i node p med et nytt subtre t */
    Tree replaceSubtree(Position p, Tree t)   }
```

BinaryTree: likhet vs. isomorfisme







I-120 bok

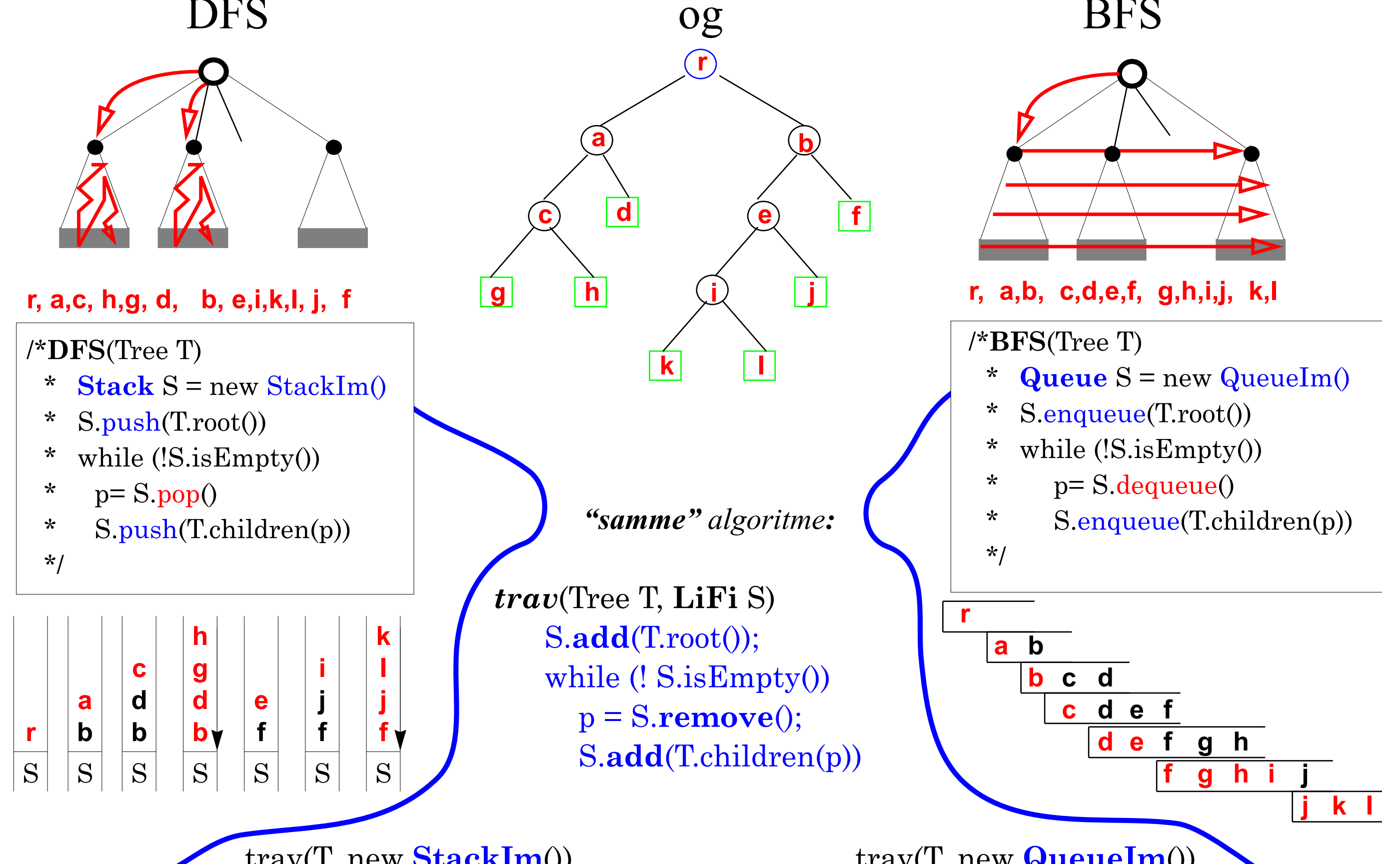
Chap. I Design Principles
- **1.1** DS & Alg
- **1.2** OO-Programming
- **1.3** JAVA

Chap. II Analysis Tools
- **2.1** Alg. Analysis
- **2.2** Running Time
  - **2.2.1** O-notation
  - **2.2.2** Avarage Case
- **2.3** Worst Case

Chap. III Basic DS
- **3.1** Stacks
- **3.2** Queues

I-120 bok

Chap. I Design Principles
Chap. II Analysis Tools
Chap. III Basic DS
- **1.1** DS & Alg
- **1.2** OO-Programming
- **1.3** JAVA
- **2.1** Alg. Analysis
- **2.2** Running Time
- **2.3** Worst Case
- **3.1** Stacks
- **3.2** Queues
  - **2.2.1** O-notation
  - **2.2.2** Avarage Case



PreOrder

gjør jobben
FØR
rekursive kall

```
DFS(Tree T, Position v)
  Print(v.element())
  for hver p i T.children(v)
     DFS(T,p)
```

I-120 bok

- **Chap. I Design Principles**
  - **1.1** DS & Alg
  - **1.2** OO-Programming
  - **1.3** JAVA
- **Chap. II Analysis Tools**
  - **2.1** Alg. Analysis
  - **2.2** Running Time
    - **2.2.1** O-notation
    - **2.2.2** Avarage Case
  - **2.3** Worst Case
- **Chap. III Basic DS**
  - **3.1** Stacks
  - **3.2** Queues

DFS

i120 bok

I, II, III

1.1, 1.2, 1.3, 2.1, 2.2, 2.3, 3.1, 3.2

2.2.1, 2.2.2

PostOrder

gjør jobben
ETTER
rekursive kall

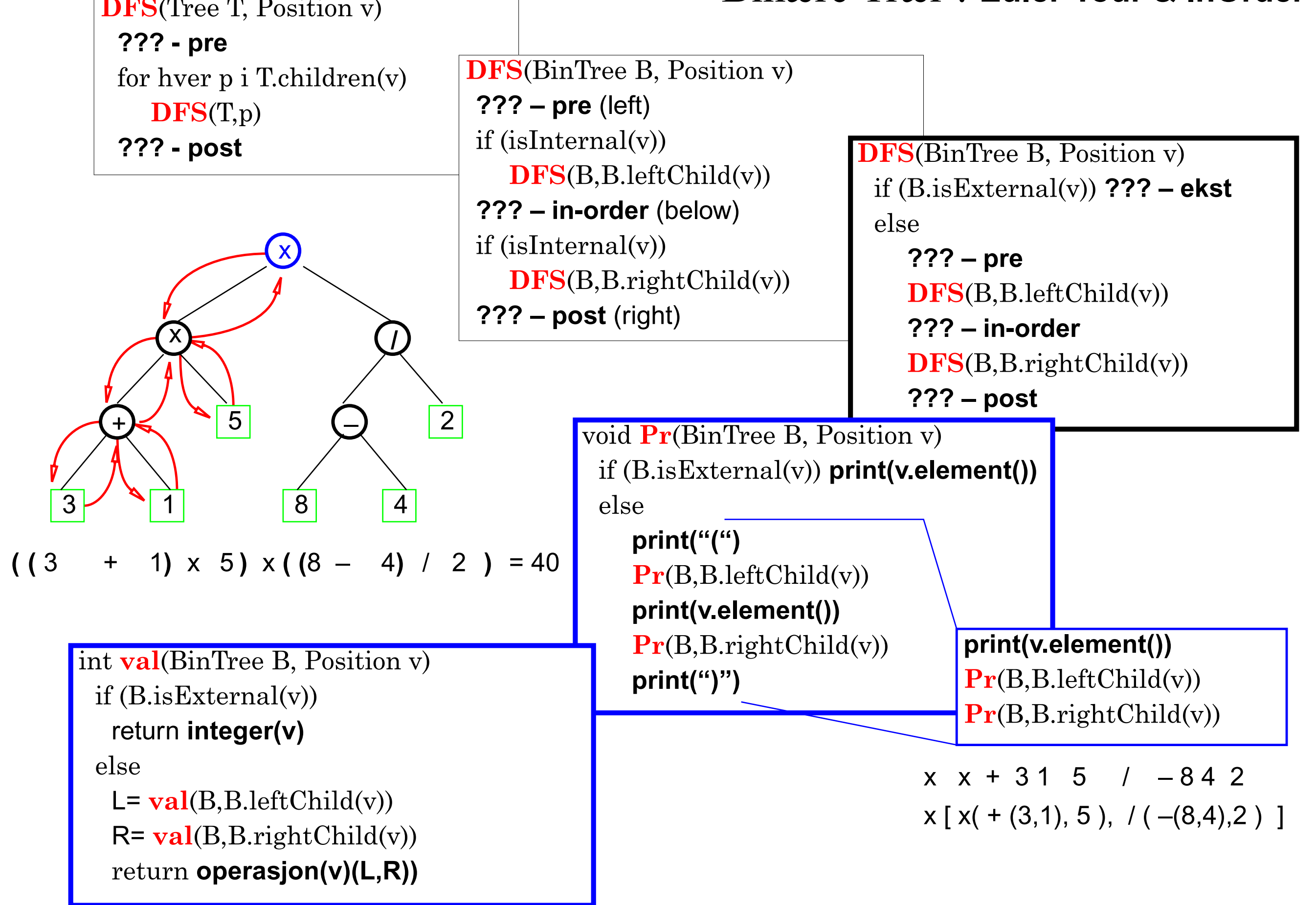
```
DFS(Tree T, Position v)

  for hver p i T.children(v)
     DFS(T,p)
  Print(v.element())
```

  - **1.1** DS & Alg
  - **1.2** OO-Programming
  - **1.3** JAVA
- **Chap. I Design Principles**
  - **2.1** Alg. Analysis
    - **2.2.1** O-notation
    - **2.2.2** Avarage Case
  - **2.2** Running Time
  - **2.3** Worst Case
- **Chap. II Analysis Tools**
  - **3.1** Stacks
  - **3.2** Queues
- **Chap. III Basic DS**

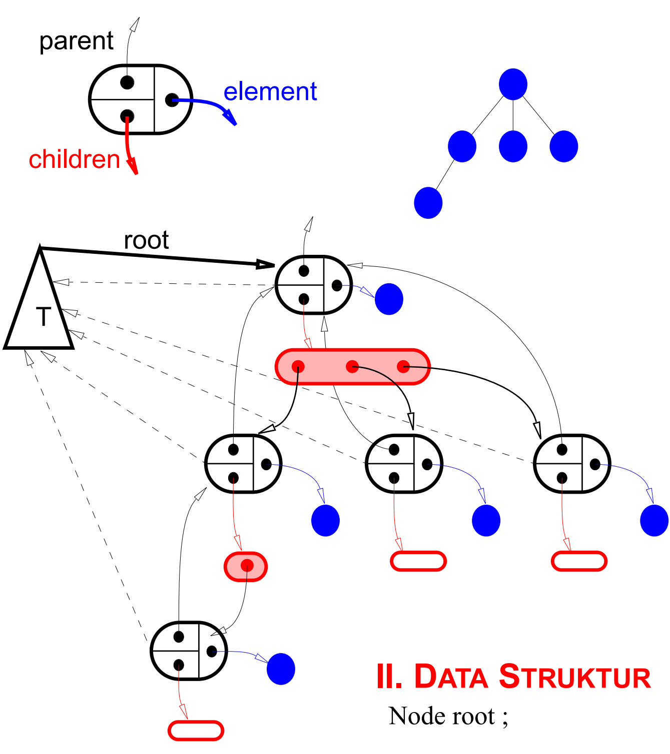
I-120 bok

Binære Trær : Euler-Tour & InOrder



Implementasjon av Tree – *med LenketStruktur*

I. Data Representasjon



II. Data Struktur

Node root ;

III. Data Invariant *: for alle Position* p, v *i* T:

- p.container() == T • T.root() != null
- isRoot(p) ↔ (p.parent()==null)
- T.isEmpty() ↔ T.root().elem() == null
- isExternal(p) == p.children().isEmpty()
- isInternal(p) == !p.children().isEmpty()
- v.parent()==p ↔ v er bland p.children()
- p.children() != null

```
public class Node implements Position
{ private Object elem; private Node parent;
  private Container  cont;  private Sequence children;
  public Node(Object e, Node p, Container c) {
    setElement(e); setParent(p);  setContainer(c);
    children= new SequenceLL(); }
  public Object element() { return elem; }
  public void setElement(Object e) { elem= e; }
  public Container container() { return cont;}
  public void setContainer(Container c) { cont= c;
    for alle x i children() : x.setContainer(c); }
  public Node parent() { return parent; }
  public void setParent(Node p) { parent= p; }
  public Container children() { return children; }
  public void setChildren(Container c) { children= c; }
```

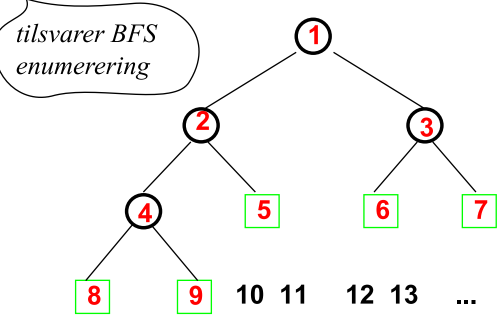

Sequence implementasjon av BinaryTree ADT

I. Data Representasjon

for en Position v i treet T, la sn(v) være et tall gitt ved:

- hvis T.isRoot(v) så sn(v) = 1
- hvis T.leftChild(v) == u så sn(u) = 2*sn(v)
- hvis T.rightChild(v) == u så sn(u) = 2*sn(v) + 1

sn *bestemmer nodens stilling i sekvensen*

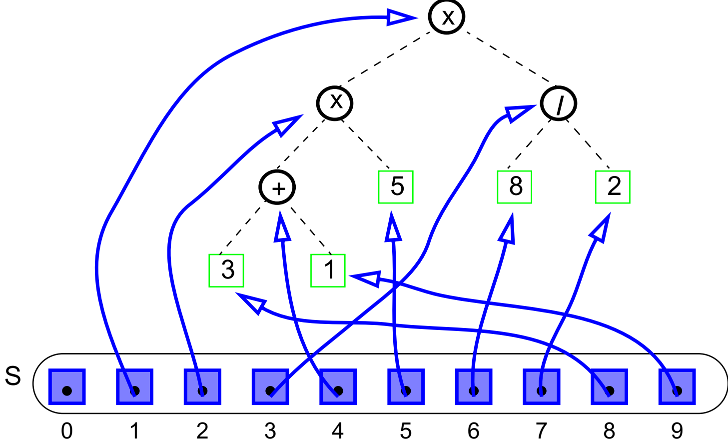


III. DataInvariant :

- S.atRank[1] == root
- S.atRank[2*v] == T.leftChild(S.atRank[v])
- S.atRank[2*v+1] == T.rightChild(S.atRank[v])

II. Data Struktur

Sequence S // *Sekvens-Position er Tree-Position*




```
Position leftChild(Position p) {
   return S . atRank( S.rankOf(p)*2 );}
Position root() { return S . atRank(1); }
```

- *alle* BinaryTree *operasjoner (unntatt positions(), elements()) er* O(1) ***relativt*** *til* Sequence
- *Spesielt adekvat for "**komplette**" binære trær*

Oppsummering

1. Trær og Binære Trær:

- *definisjoner og terminologi*
- *egenskaper*

2. Tre-algoritmer – traversering:

- *DFS og BFS*
- *DFS :*
  - *– pre– og postorder,*
  - *– inorder for BinaryTree*

3. Tree og BinaryTree ADT

4. Implementasjon av trær:

- *LenketStruktur*
- *Sekvens – BinaryTree*
- *kompleksitet*