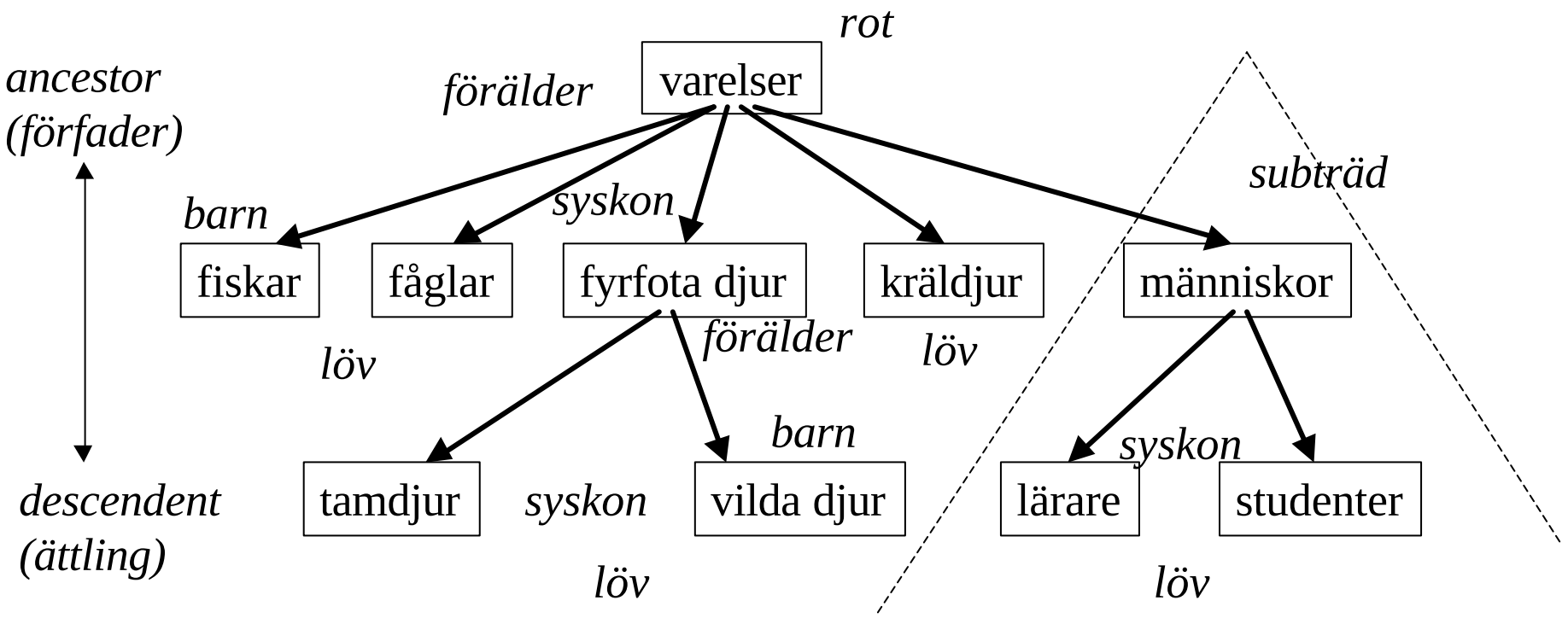
Föreläsning 10

Träd - C&P kap. 10
speciellt binära sökträd sid. 452

Dessa bilder finns i PDF-format på
http://dsv.su.se/courses/pm2/f10/index.html

Träd allmänt

Binär-länkad (icke-linjär), hierarkisk struktur. Består av noder förbundna med kanter. Varje nod har exakt en förälder utom roten som har ingen. Noder utan barn kallas löv.

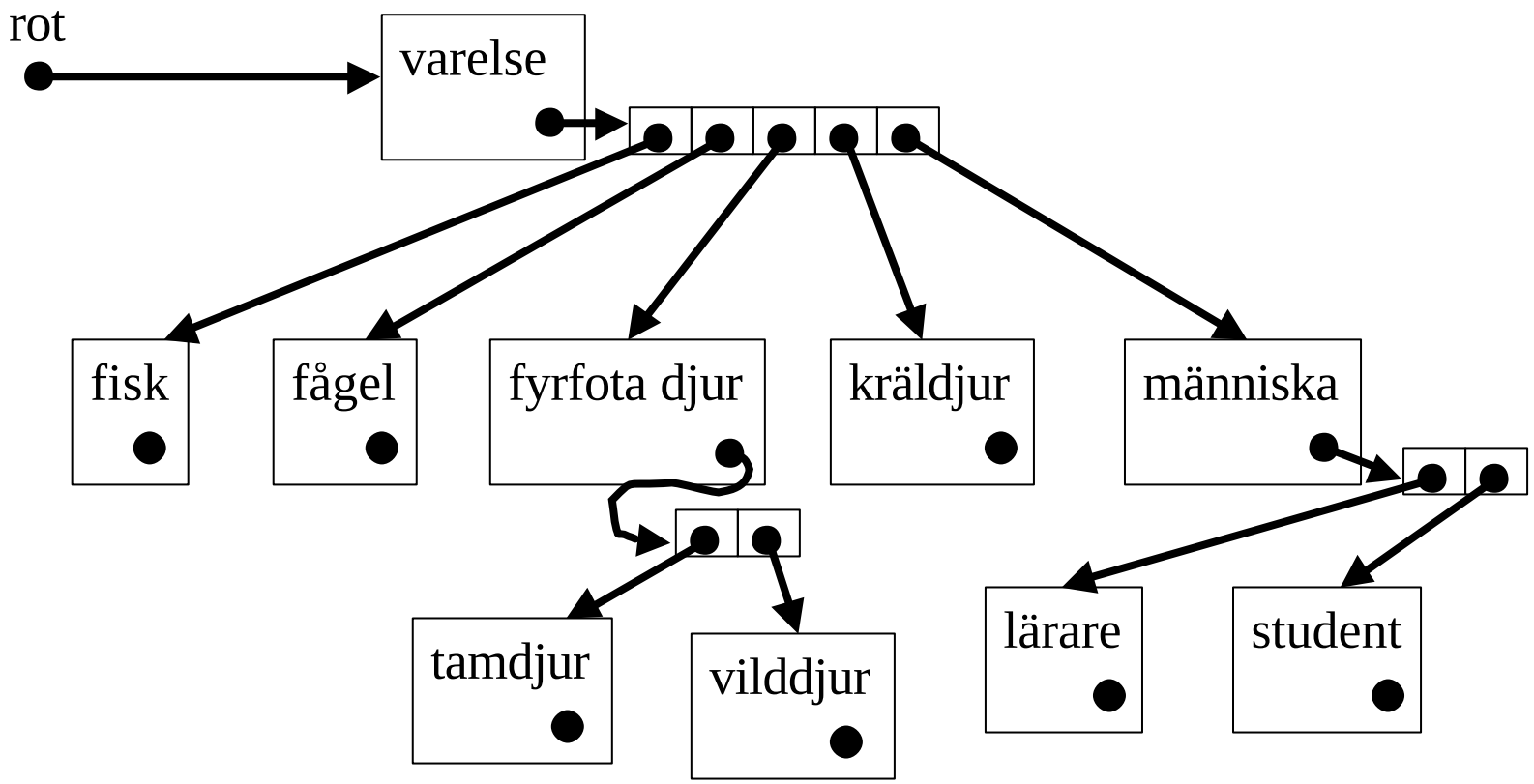


en nods nivå = antalet steg till roten
trädets höjd = maximala nodnivån

rotens nivå = 1
tomt träds höjd = 0

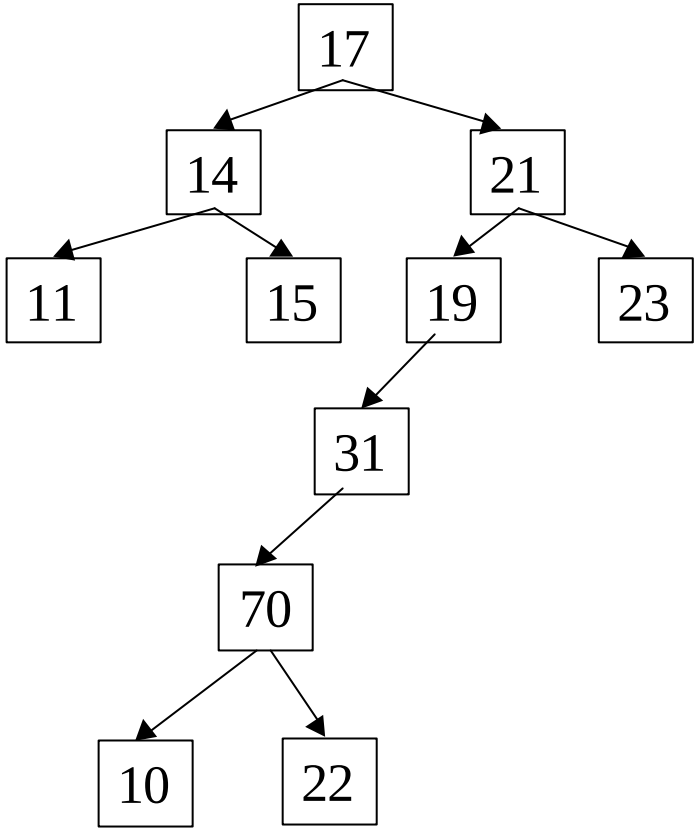
Generella träd - en möjlig implementering

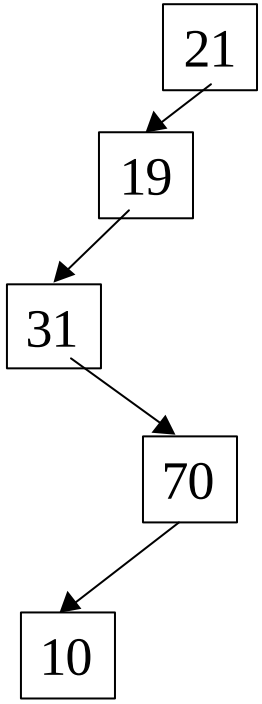
Varje nod har en array med referenser till sina barn
(alternativt kan varje nod ha en länkad lista med referenser till barnen)



Binära träd

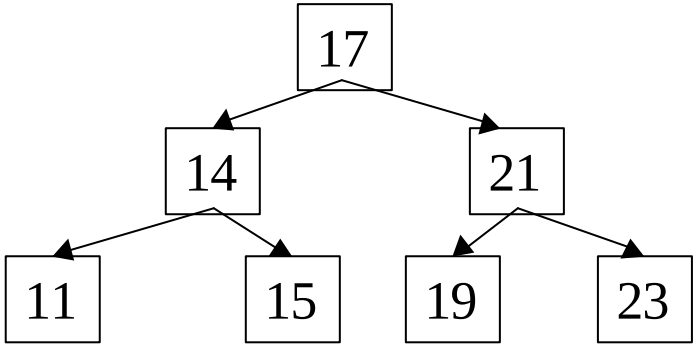
Maximal utgreningsgrad = 2, dvs varje nod har högst 2 barn
Brukar kallas vänster- resp. högerbarn (-subträd).



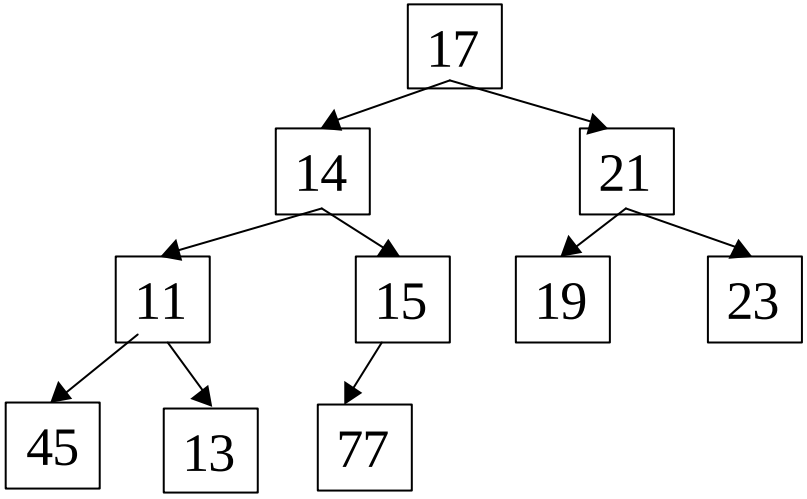


Binära träd

Fullständigt binärt träd: alla påbörjade nivåer är helt fyllda,
dvs varje löv har samma nivå och varje icke-löv har två barn

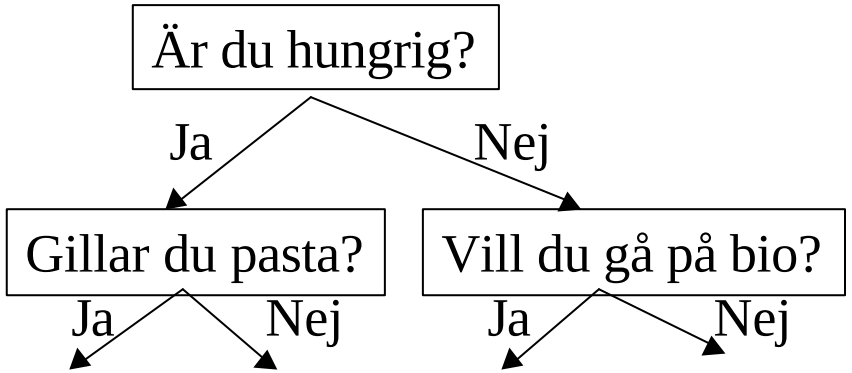


Komplett binärt träd: alla nivåer utom det nedersta är helt fyllda,
på den nedersta nivån ligger noderna till vänster



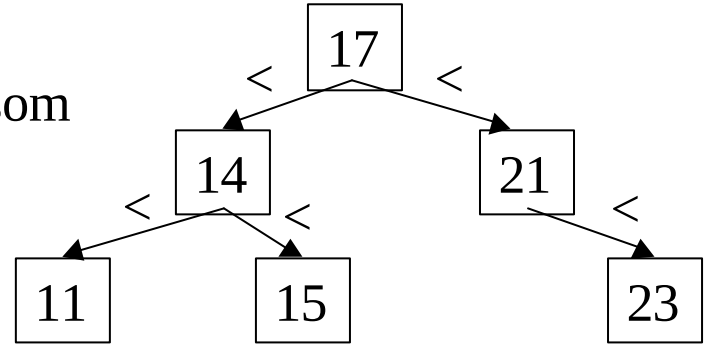
Binära träd - användningsexempel

Beslutsträd



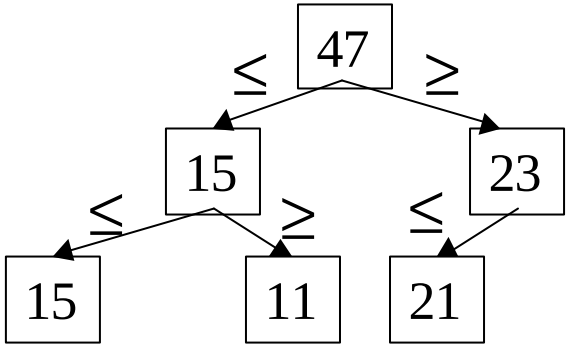
Binära sökträd

- noder i vänstra subträdet har värden som är mindre än eller lika med förälderns,
- noder i högra subträdet har större värden än föräldern

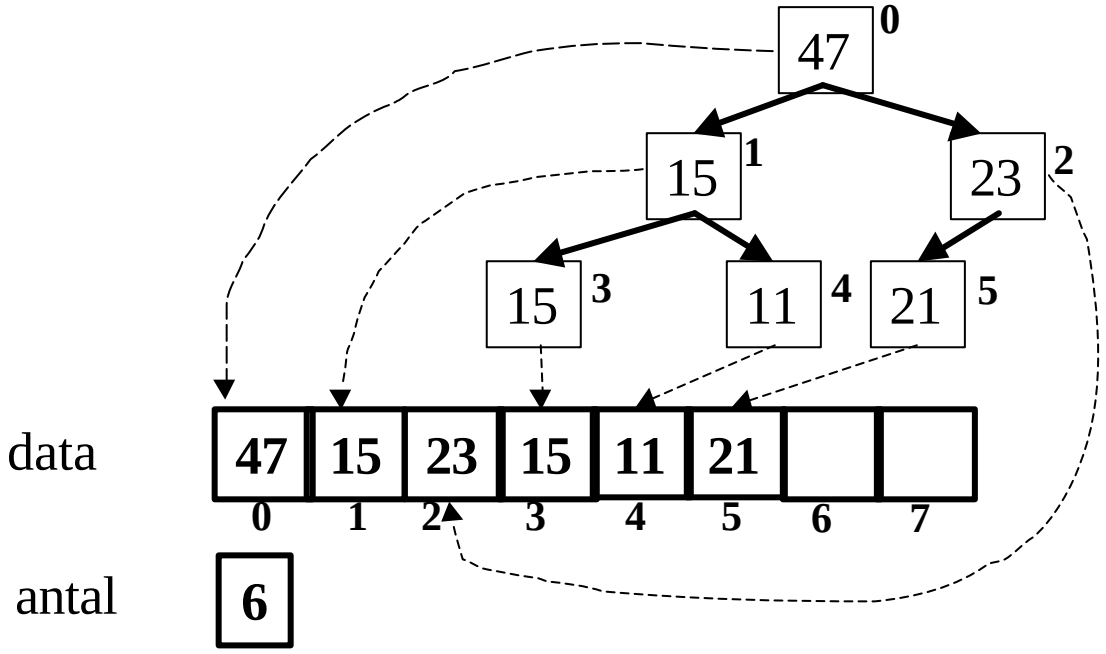


Heap

- barnens noder har värden som är lägre än eller lika med förälderns,
- måste vara komplett



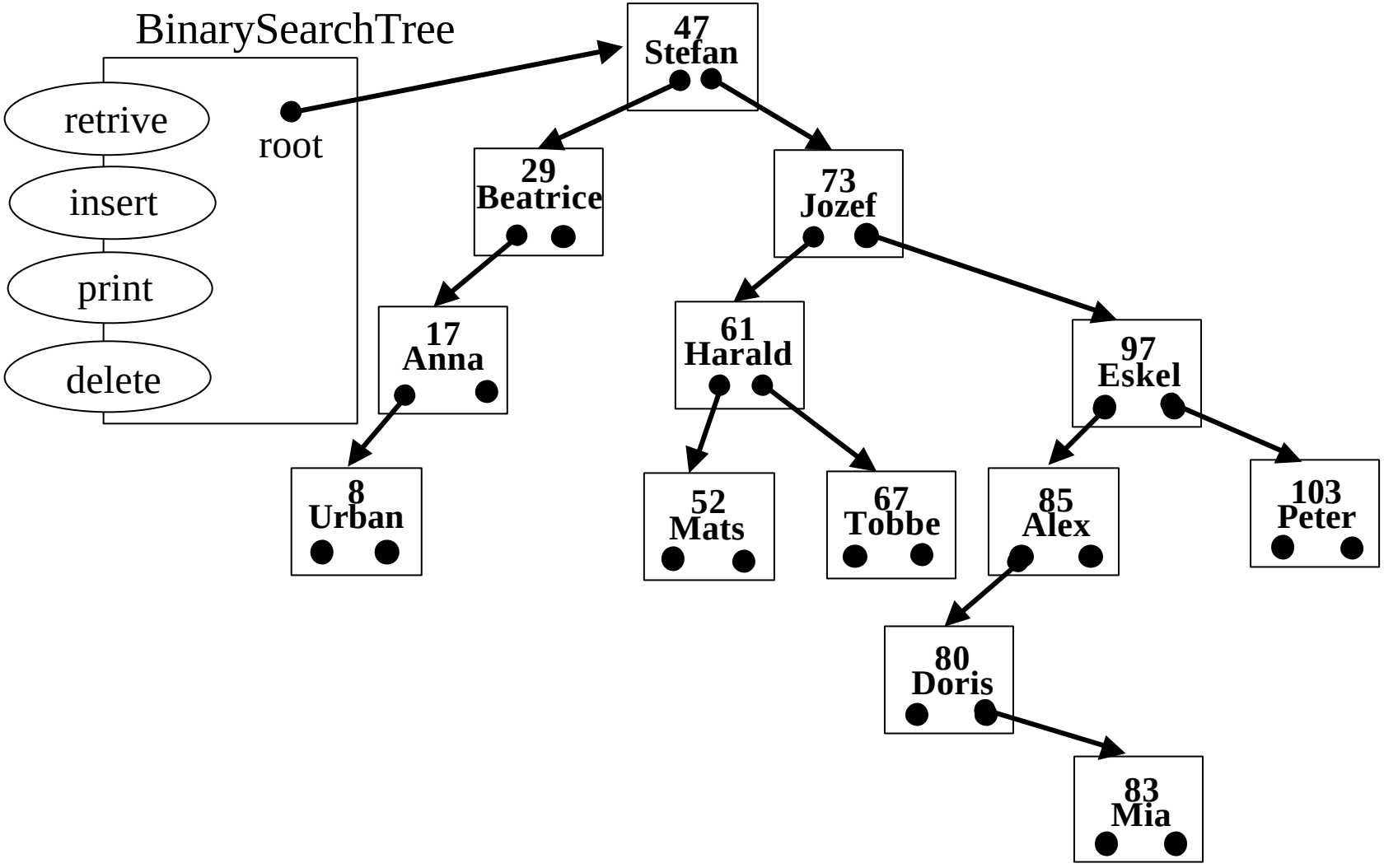
Arrayimplementering av kompletta binära träd



roten finns i **`data[0]`**
i-te nodens vänstra barn finns i **`data[2*i+1]`** (om **`2*i+1 < antal`**)
i-te nodens högra barn finns i **`data[2*i+2]`** (om **`2*i+2 < antal`**)
i-te nodens förälder finns i **`data[(i-1)/2]`** (om i > 0)
(med i-te nod menar jag noden i **`data[i]`**)

Länkad implementering av binära träd

Exempel: binärt sökträd



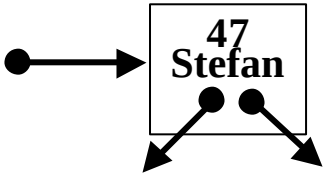
Rekursiv definition av binära träd

Ett binärt träd är

• tomt

eller

• består av en nod och dess vänstersubträd och dess högersubträd och båda subträden är binära träd



Nodklass för ett binärt träd (ej återanvändbar)

```
class TNode{
    int id;
    String data;
    TNode left, right;

    TNode(int i, String d){
      id=i;
      data=d;
      left=right=null;
    } // Konstruktor

    public String toString(){
      return id + ": " + data;
    } // toString
} // TNode
```

Dataattributen är inte skyddade för att få tydligare kod i metod-exemplen, de borde givetvis vara deklarerade som **`private`** och kunna avläsas med **`get`**-metoder

Klassen **BinSearchTree** (ej återanvändbar)

```
class BinSearchTree {
    private TNode root=null;

    public TNode retrive(int sought){
      ...
    }

    public void insert(int id, String data){
      ...
    }

    public void print(){
      ...
    }

    void delete(int sought){
      ...
    }
} // BinSearchTree
```

Rekursiv sökfunktion

```
class BinSearchTree {
    private TNode root=null;

    TNode retriveNode(TNode node, int sought){
      if (node == null)
          return null;
      else if (node.id == sought)
          return node;
      else if (node.id > sought)
          return retriveNode(node.left, sought);
      else
          return retriveNode(node.right, sought);
    } // retriveNode
```

Gränssnittsmetoder och privata metoder

Antag deklarationen **`BinSearchTree tree = new BinSearchTree();`**
Antag även att några noder har skapats och lagts till trädet **`tree`**.
Ett anrop av den rekursiva metoden **`retriveNode`** skulle behöva se ut så här:

Tnode tn = tree.retriveNode(tree.root, 73);

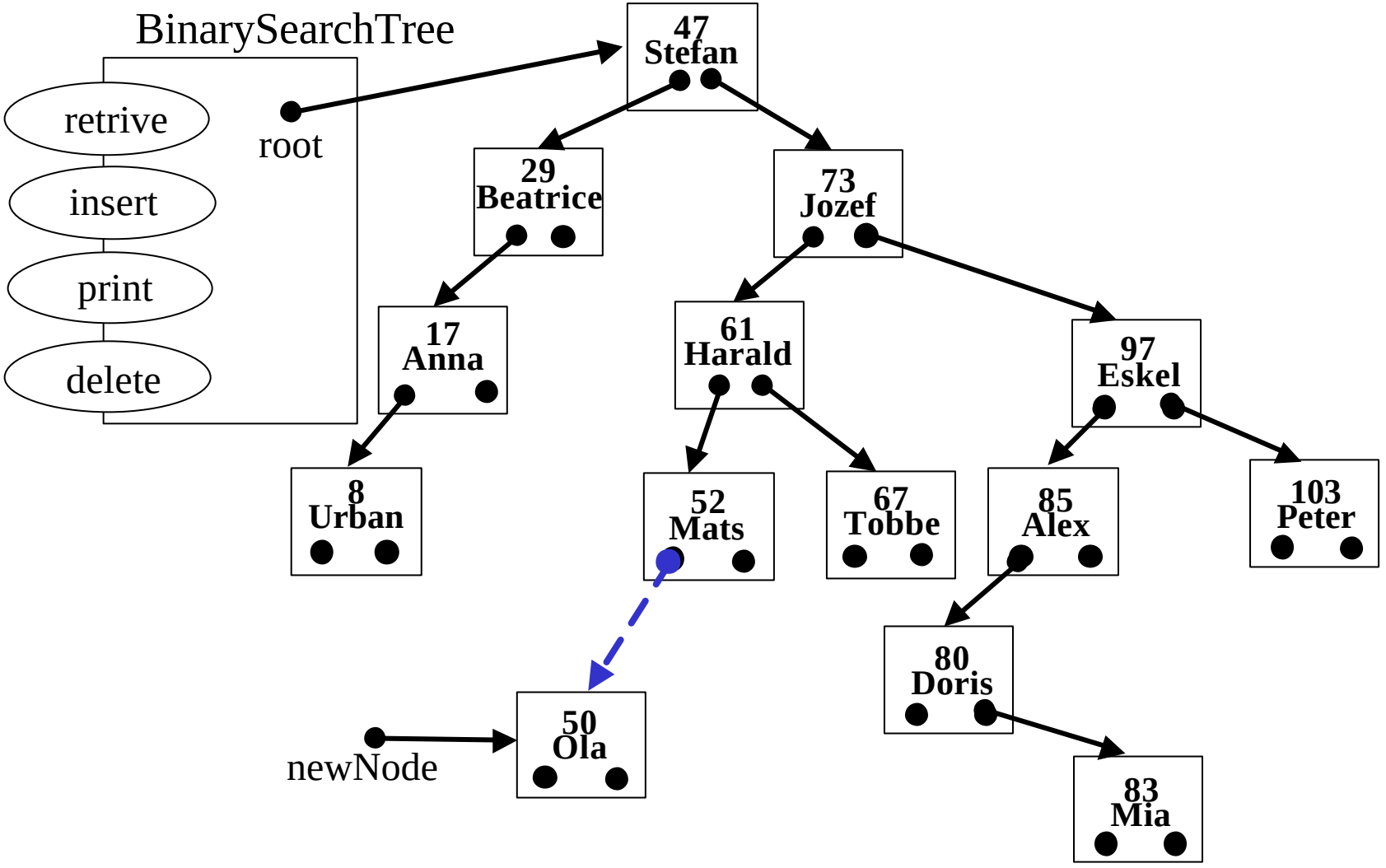
Men en tillämpning har inte åtkomst till **`tree.root`**, dessutom känns det ondödigt att behöva ange **`tree.root`** vid varje anrop av metoden.
Man brukar därför skapa publika gränssnittsmetoder som bara är till för att starta rekursionen och privata rekursiva metoder som gör jobbet:

```
class BinSearchTree {
    private TNode root=null;

    public TNode retrive(int sought){
      return retriveNode(root, sought);
    }

    private TNode retriveNode(TNode node, int sought){
      // som tidigare
    }
```

Insättning av ny nod i trädet



Kod för insättning av ny nod i trädet

```
class BinSearchTree {
    private TNode root=null;

    public void insert(int id, String data){
      TNode newNode = new TNode(id, data);
      root=insertNode(root, newNode);
    }

    private TNode insertNode(TNode node, TNode newNode){
      if (node == null)
          node = newNode;
      else if (node.id > newNode.id)
          node.left = insertNode(node.left, newNode);
      else
          node.right = insertNode(node.right, newNode);
      return node;
    }
```

Traversering (genomgång) av trädet

Exempel: utskrift i sorteringsordning

```
class BinSearchTree {
    private TNode root=null;

    public void print(){
      printTree(root);
    }

    private void printTree(TNode node){
      if (node != null){
          printTree(node.left);
          System.out.println(node);
          printTree(node.right);
      }
    }
```

Träd - traverseringsordning

- inorder - vänstra subträdet, noden själv, högra subträdet
  I binära sökträd ger detta traversering i sorteringsordning

- preorder - noden själv, vänstra subträdet, högra subträdet

```
private void postorderPrint(TNode node){
      if (node != null){
          System.out.println(node);
          printTree(node.left);
          printTree(node.right);
      }
}
```

- postorder - vänstra subträdet, högra subträdet, noden själv

Borttagning av en nod från trädet

```
private TNode deleteNode(TNode node, int sought)
        throws TreeException {
   if (node == null)
       throw new TreeException(”No such element!”);
   else if (node.id > sought)
       node.left = deleteNode(node.left, sought);
   else if (node.id < sought)
       node.right = deleteNode(node.right, sought);
   else if (node.left == null)
       node = node.right;
   else if (node.right == null)
       node = node.left;
   else {
       TNode tmp=findLeftmost(node.right);
       node.id = tmp.id;
       node.data = tmp.data;
       node.right = deleteLeftmost(node.right);
   }
   return node;
 }
```

Borttagning av en nod från trädet (forts.)

```
public void delete(int sought){
      root = deleteNode(root, sought);
}

private TNode findLeftmost(TNode node){
      if (node.left == null)
          return node;
      else
          return findLeftmost(node.left);
}

private TNode deleteLeftmost(TNode node){
      if (node.left == null)
          return node.right;
      else {
          node.left = deleteLeftmost(node.left);
          return node;
      }
}
```