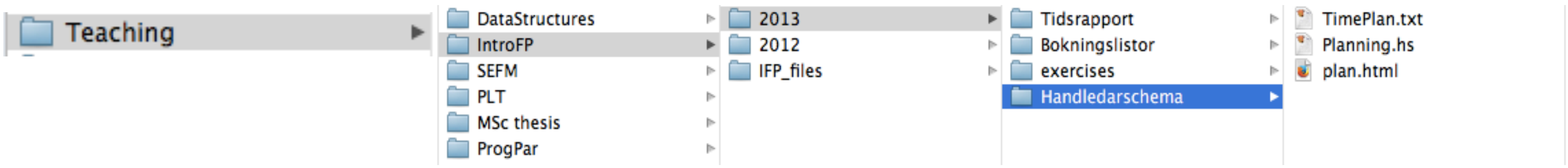
# Lecture 8
# Data Structures (DAT037)

Ramona Enache
(with slides from Nick Smallbone and Nils Anders Danielsson)

# Trees

A **tree** is a hierarchical data structure
Each node can have several children but only has one parent
The root has no parents; there is only one root

Example: directory hierarchy



# Trees (as graphs)

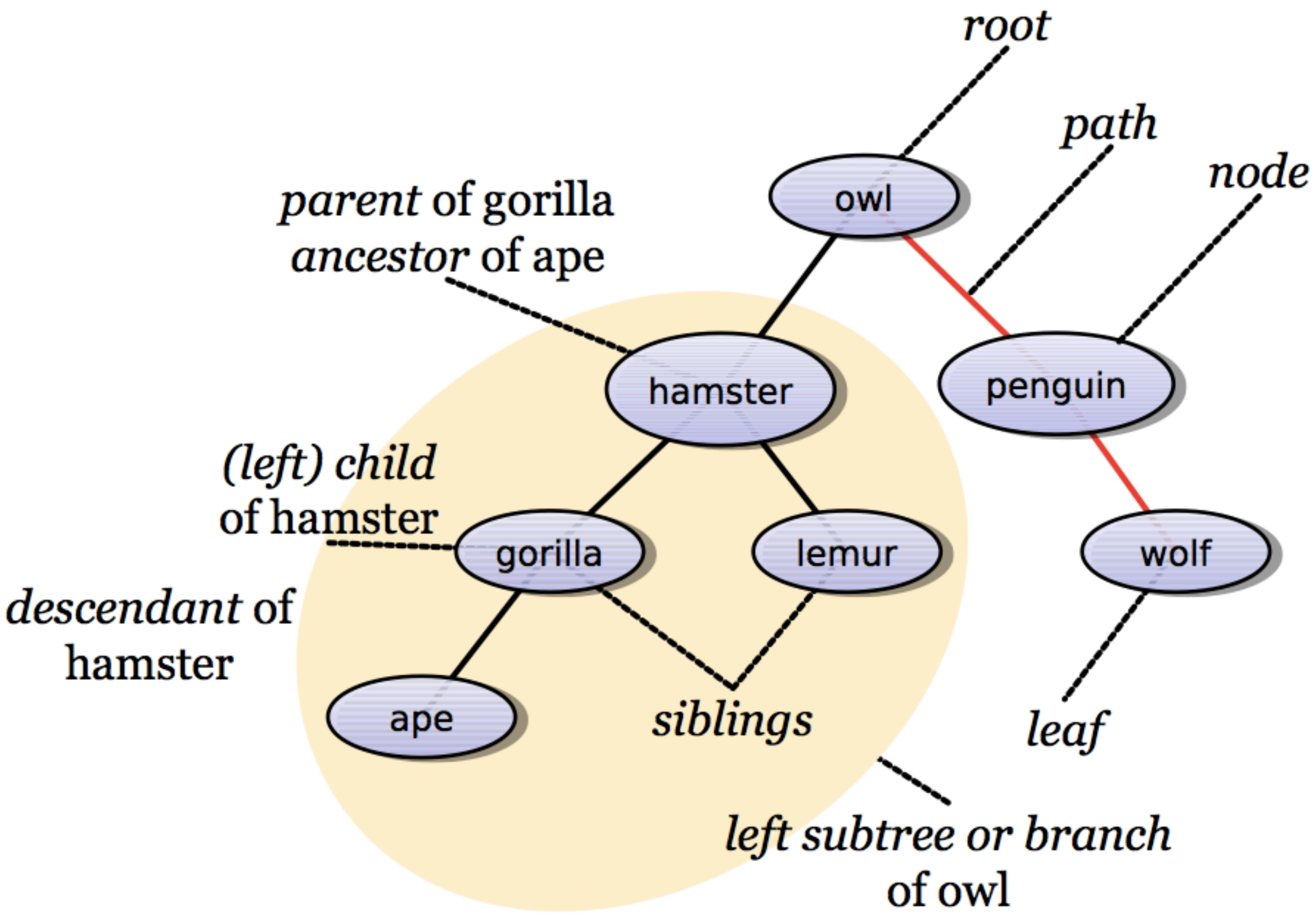
**Also**

A **tree** is

+ undirected, unweighted, simple (not multi)

acyclic graph

# Tree Terminology



# Tree Terminology

The depth of a node is the distance from the root
The height of a tree is the number of levels in the tree
The size of a tree is the number of nodes in it

# Binary Trees

A **binary tree** is a tree with at most two children for each node

**Java**:

```
class Tree<A> {
    class TreeNode {
        A contents;
        TreeNode left, right; }

    TreeNode root;}
```

Can be null if child is missing

Can be null if tree is empty

**Haskell**:

```
data Tree a = Node a (Tree a) (Tree a) | Empty
```

# Height of (Binary) Trees

Height:

- Empty trees have height -1.
- Otherwise: Number of steps from root to deepest leaf.

**Haskell**:

```
height :: Tree a -> Integer
height Empty = -1
height (Node _ l r) = 1 + max (height l) (height
r)
```

# Height of (Binary) Trees

Height:

- Empty trees have height -1.
- Otherwise: Number of steps from root to deepest leaf.

**Java**:

```
int height(TreeNode n) {
  if (n == null) return -1;
   else
       return 1 +
Math.max(height(n.left),height(n.right));}
```

## Question

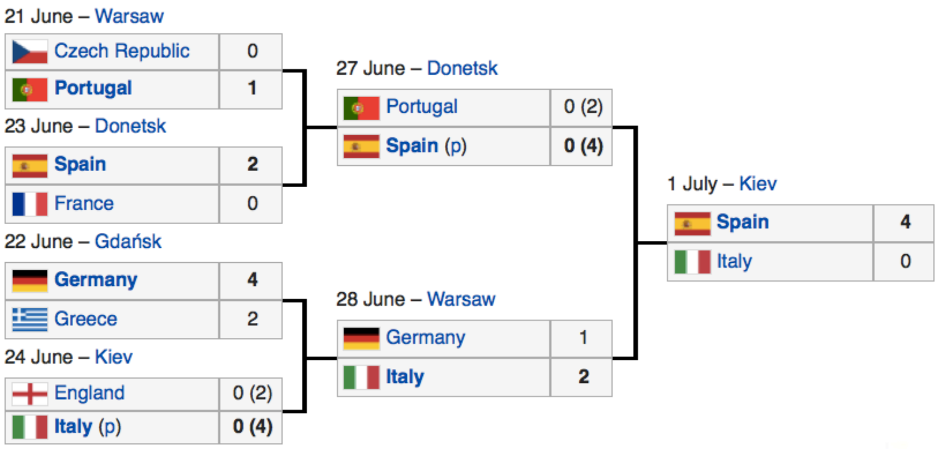
What is the smallest and largest number of nodes for a binary tree of height *n* (computed with the function define before)?

1. n+1 and 2^n-1
2. n and 2^n – 1
3. n+1 and 2^(n+1)-1
4. none of the above

govote.at
Code 882001

# Balanced Binary Trees

A tree can be balanced or unbalanced



21 June – Warsaw

| | |
|---|---|
| Czech Republic | 0 |
| **Portugal** | **1** |

23 June – Donetsk

| | |
|---|---|
| **Spain** | **2** |
| France | 0 |

22 June – Gdańsk

| | |
|---|---|
| **Germany** | **4** |
| Greece | 2 |

24 June – Kiev

| | |
|---|---|
| England | 0 (2) |
| **Italy** (p) | **0 (4)** |

27 June – Donetsk

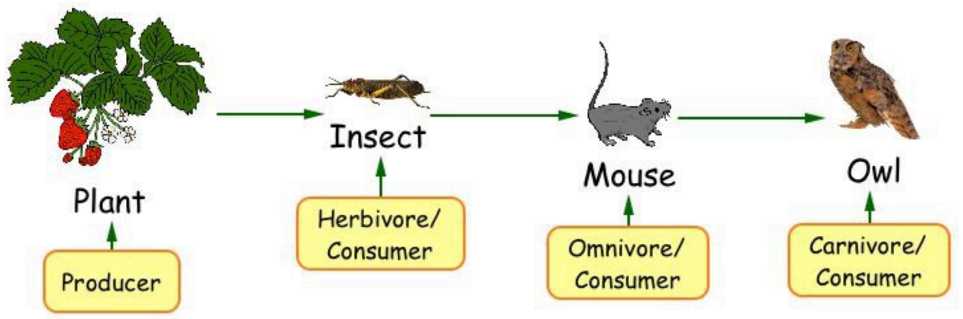
| | |
|---|---|
| Portugal | 0 (2) |
| **Spain** (p) | **0 (4)** |

28 June – Warsaw

| | |
|---|---|
| Germany | 1 |
| **Italy** | **2** |

1 July – Kiev

| | |
|---|---|
| **Spain** | **4** |
| Italy | 0 |



# Balanced Binary Trees

A tree can be balanced or unbalanced

If a tree of size n is
- balanced, its height is O(log n)
- unbalanced, its height could be O(n)

Many tree algorithms have complexity O(height of tree), so are efficient on balanced trees and less so on unbalanced trees

Normally:
- balanced trees - good ☺
- unbalanced trees – bad ☹

# Traversal of (Binary) Trees

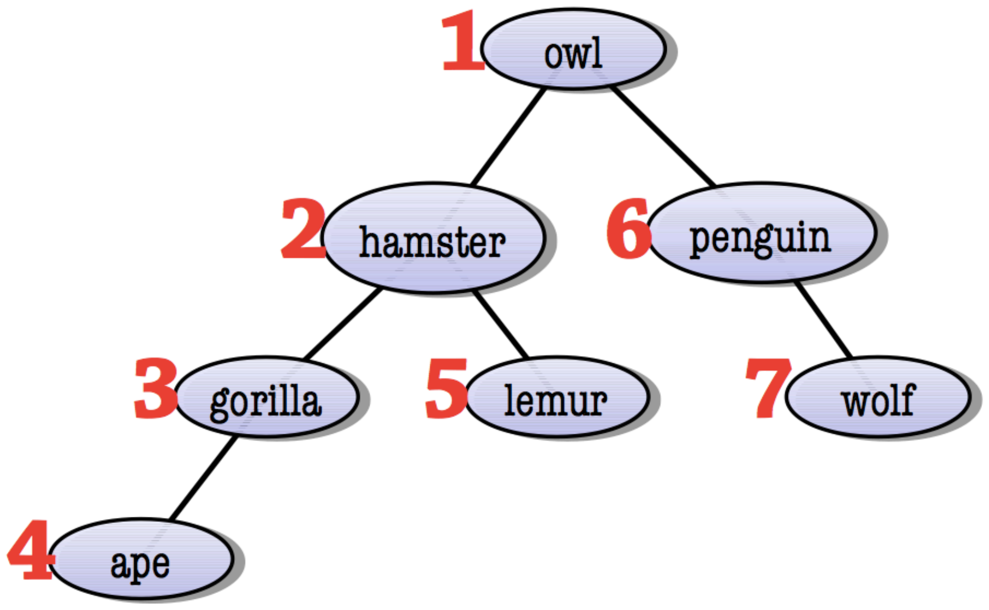
- Traversing a tree means visiting all its nodes in some order
- A traversal is a particular order that we visit the nodes in
- Four common traversals:
  - + preorder
  - + inorder
  - + postorder
  - + level-order
- For each traversal, you can define an iterator that traverses the nodes in that order

## Traversal of (Binary) Trees

- Traversing a tree means visiting all its nodes in some order
- A traversal is a particular order that we visit the nodes in
- Four common traversals:
  - + preorder
  - + inorder
  - + postorder
  - + level-order
- For each traversal, you can define an iterator that traverses the nodes in that order

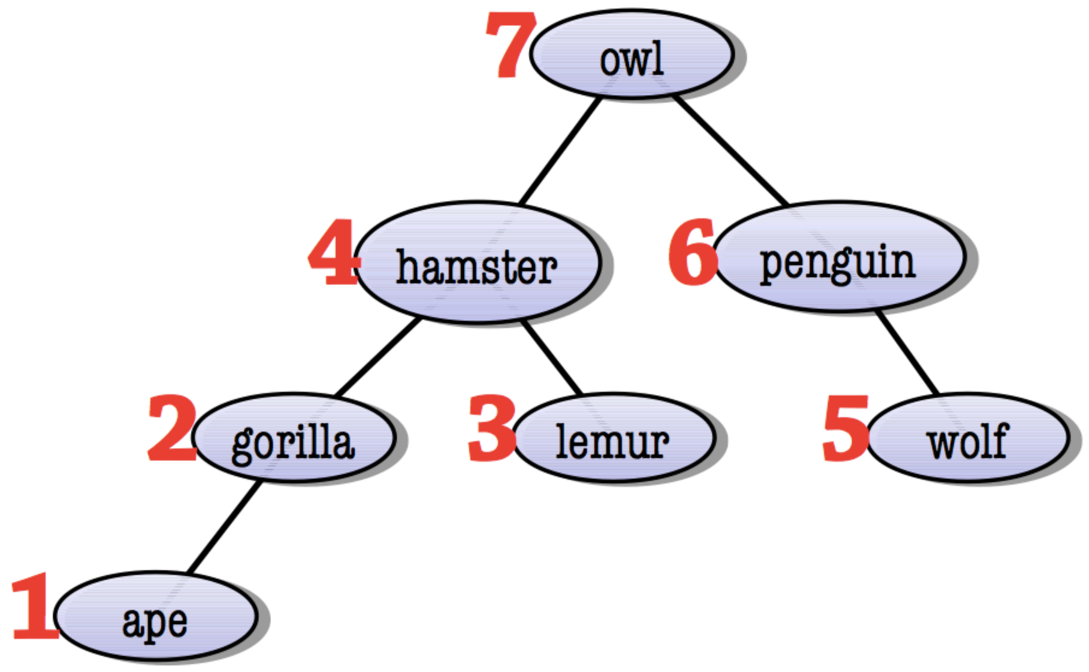
# Preorder Traversal

Visit root node, then left child, then right



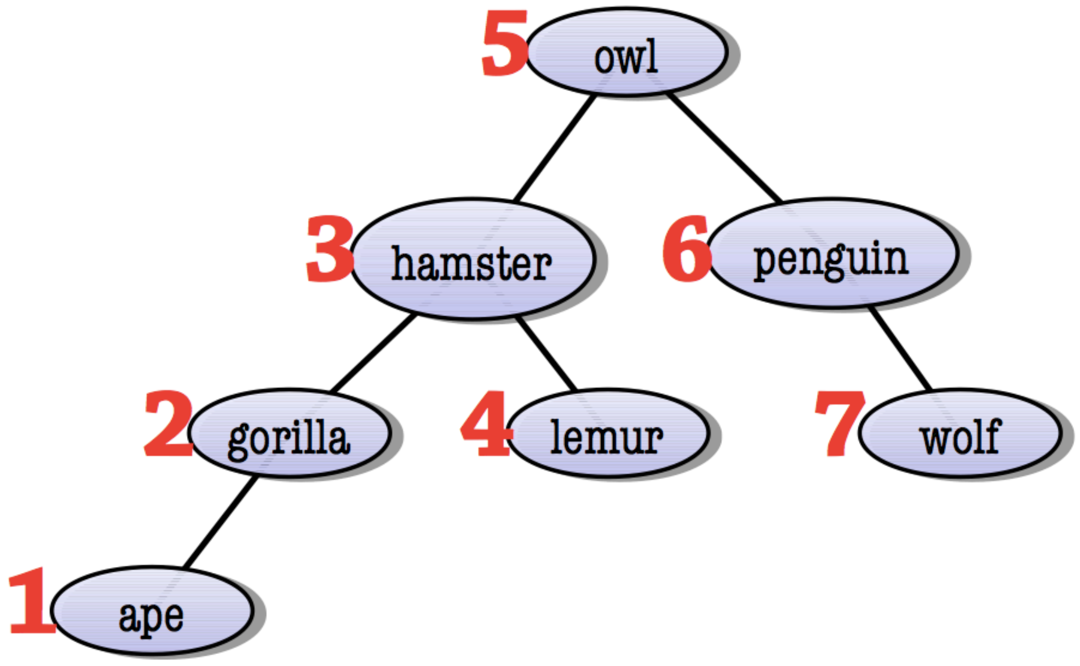
# Postorder Traversal

Visit left child, then right, then root node



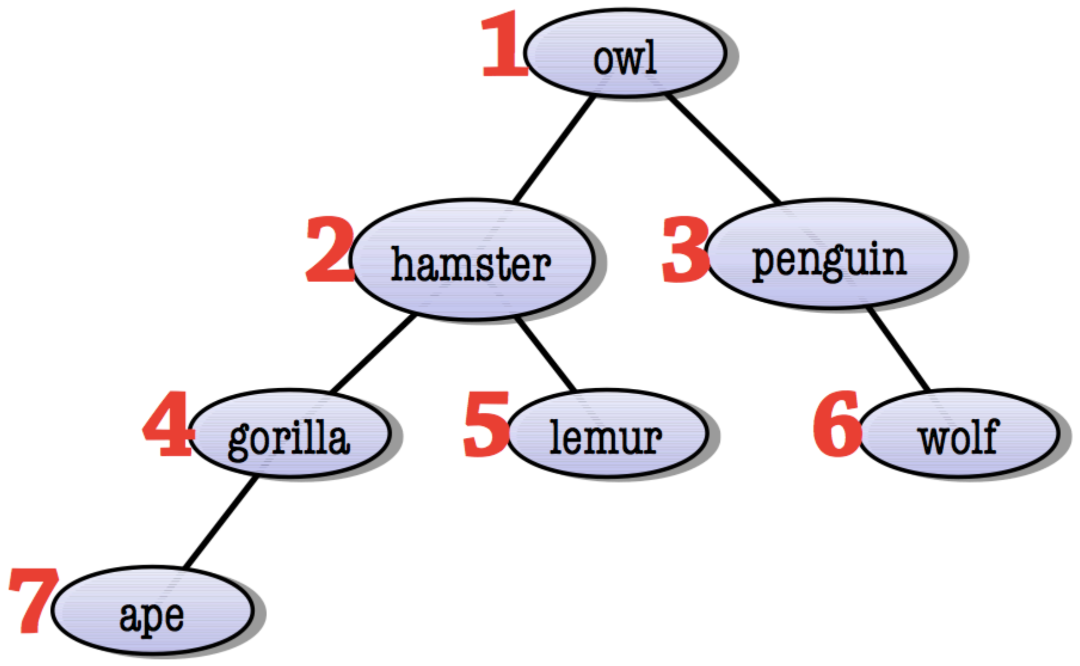
# Inorder Traversal

Visit left child, then root node, then right child



# Level-Order Traversal

Visit nodes left to right, top to bottom



# Question

Which of the tree traversal methods is equivalent to DFS ?

1. Preorder
2. Postorder
3. Inorder
4. Traversal on levels

govote.at
Code 836306

# Implementing Tree Traversals

Implement the others on your own

**Preorder**:

**Java:**

```
void preorder(Node<E> node) {
if (node == null) return;
System.out.println(node.value);
preorder(node.left); preorder(node.value);}
```

Inefficient list append

**Haskell:**

```
preorder Empty = []
preorder (Node info l r) = info : (preorder l ++
preorder r)
```

## Question

Which of the following assertions about binary trees with distinct elements holds?

1. We can recreate a tree from its preorder and postorder
2. We can recreate a tree from its inorder and preorder
3. We can recreate a tree from only one of the traversals
4. We can’t recreate a tree from any combination of two traversals

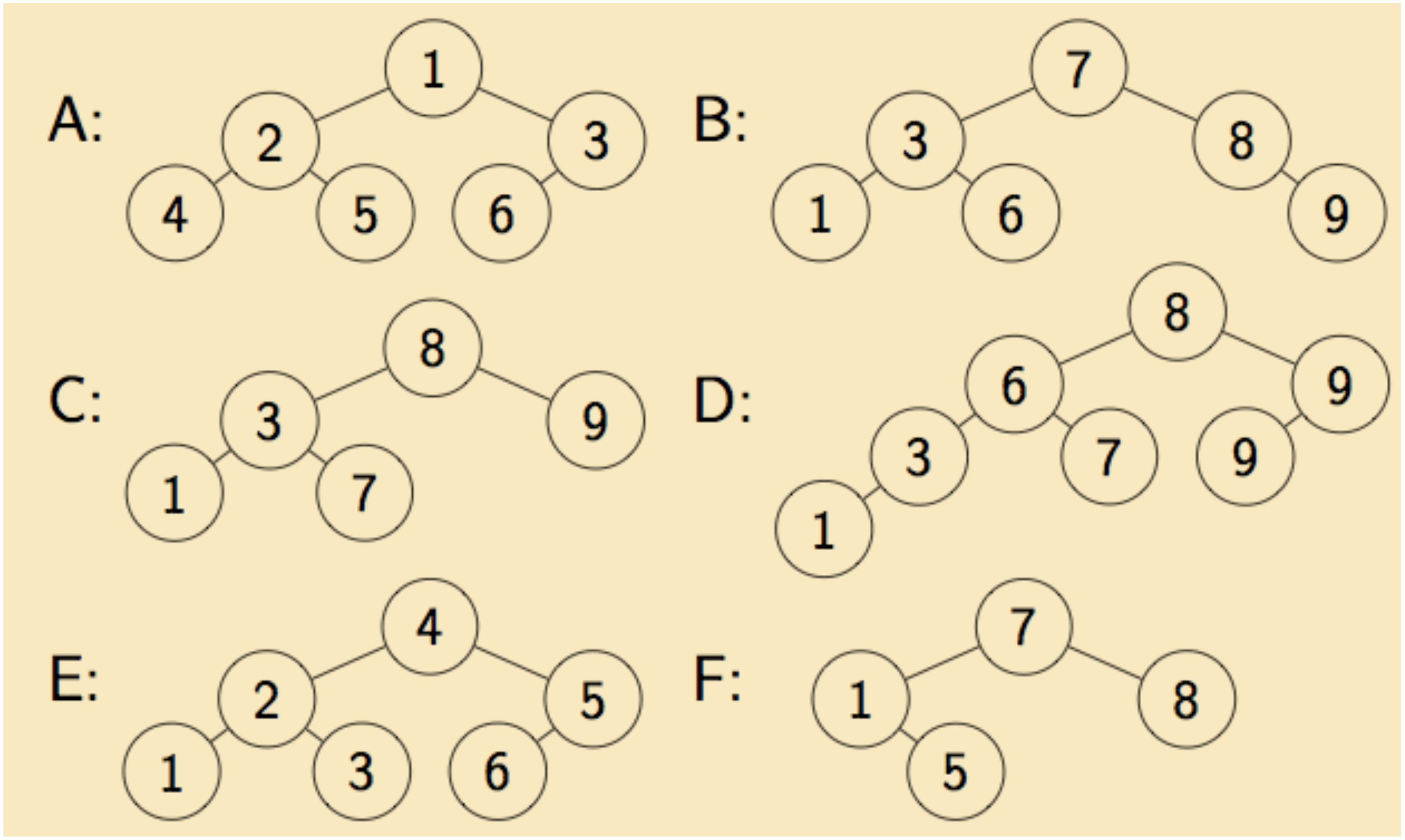
govote.at
Code 82769

## Binary Search Trees (BST)

In a **binary search tree (BST)**, every node is greater than all its left descendants, and less than all its right descendants (recall that this is an invariant)

# Binary Search Trees (BST)

Which of the following trees is a BST ?

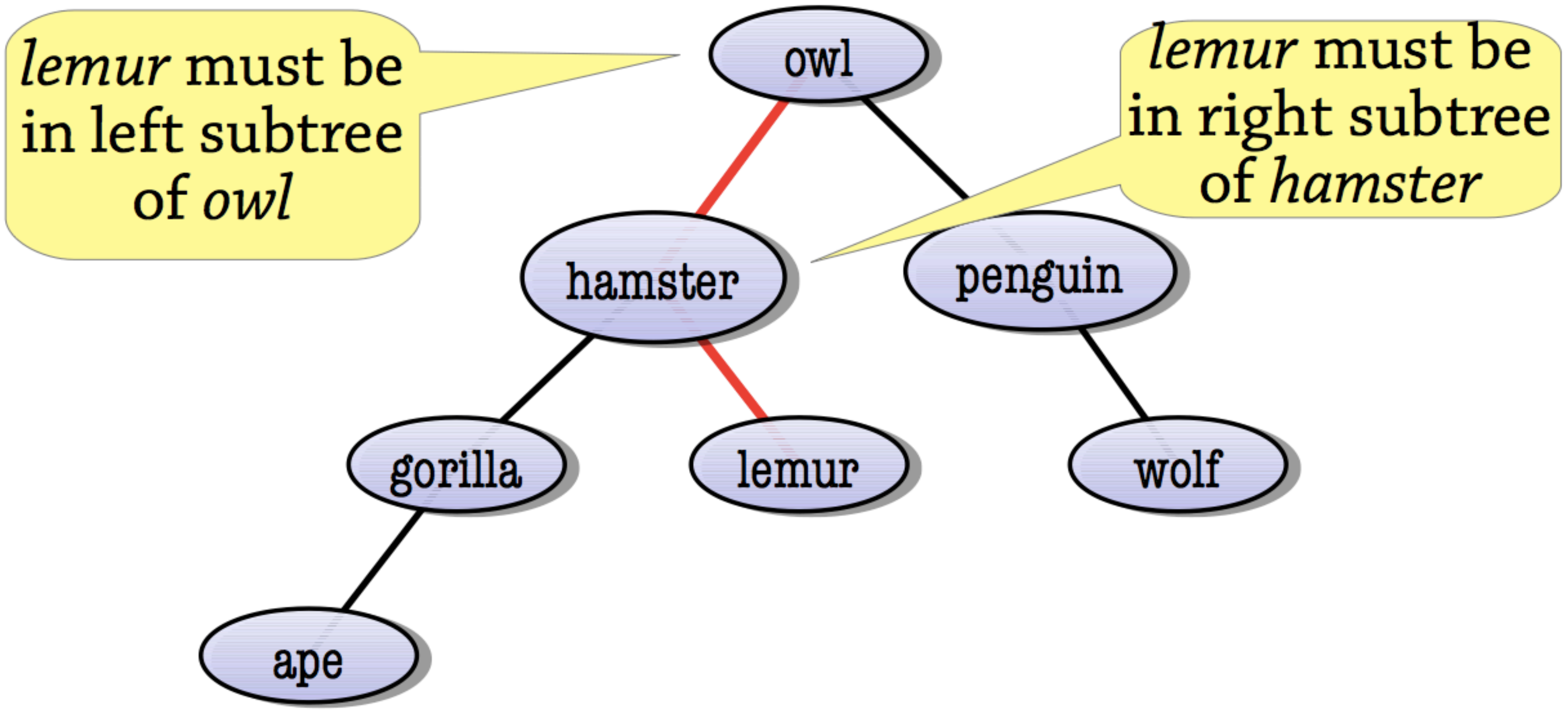


## Operations on BST

- Searching for a value
- Inserting a new value
- Deleting a value

# Searching in a BST

Finding an element in a BST is easy, because by looking at the root you can tell which subtree the element is in


*lemur* must be in left subtree of *owl*
owl
*lemur* must be in right subtree of *hamster*
hamster
penguin
gorilla
lemur
wolf
ape

## Searching in a BST

To search for target in a BST:

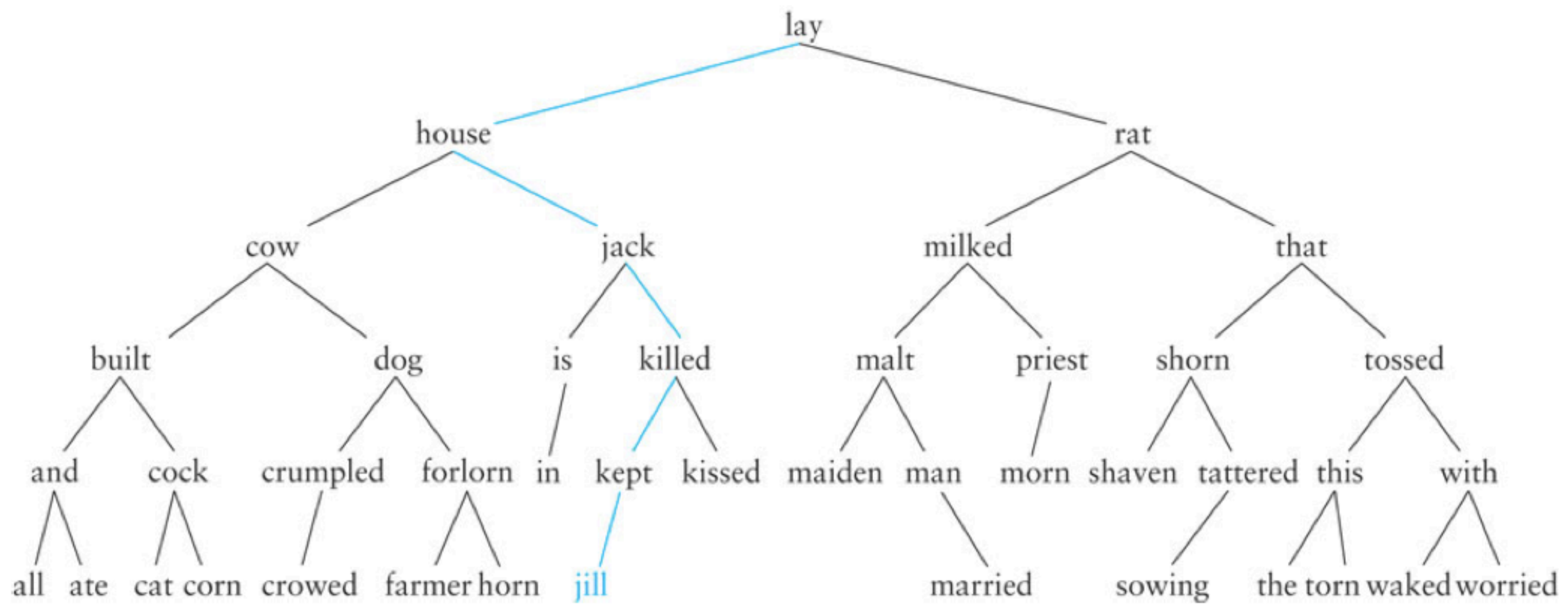
- If the target matches the root node's data, we've found it
- If the target is less than the root node's data, recursively search the left subtree
- If the target is greater than the root node's data, recursively search the right subtree
- If the tree is empty, fail

A BST can be used to implement a set, or a map from keys to values

# Inserting in a BST

To insert a value into a BST:

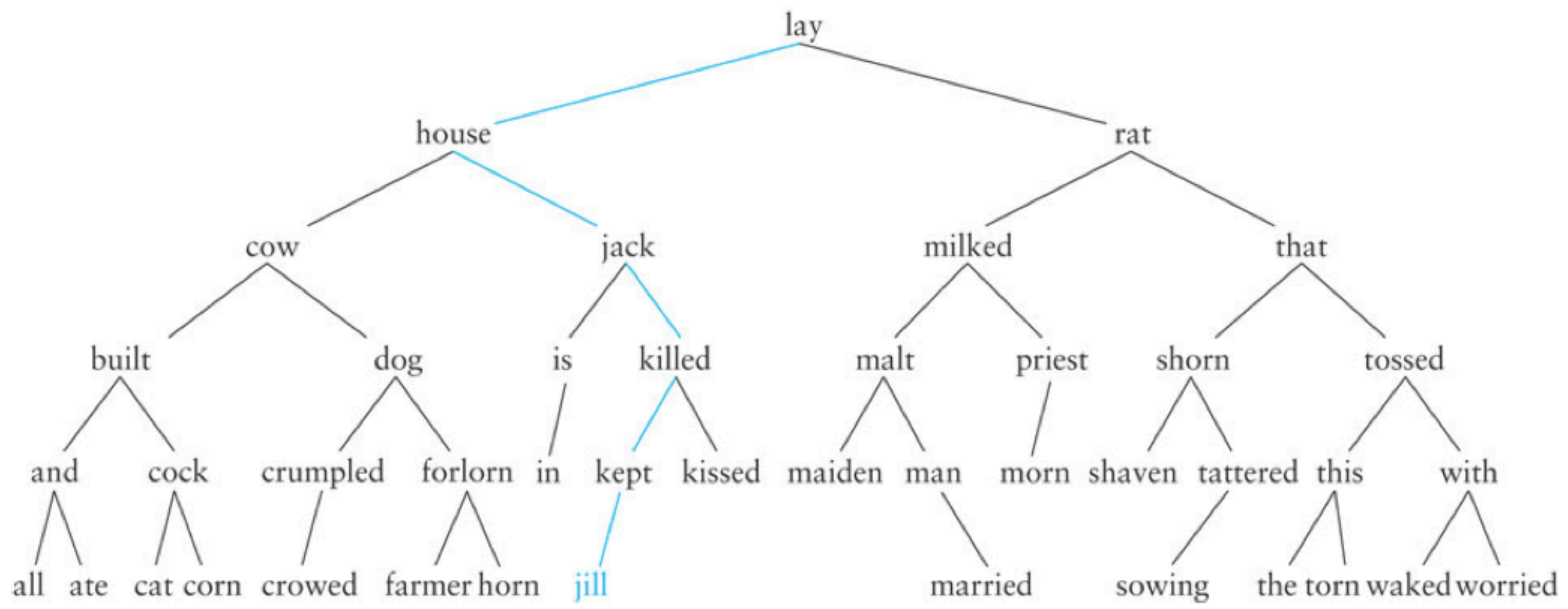
- Start by searching for the value
- But when you get to null (the empty tree), make a node for the value and place it there



# Inserting in a BST

To insert a value into a BST:

- Start by searching for the value
- But when you get to null (the empty tree), make a node for the value and place it there



## Question

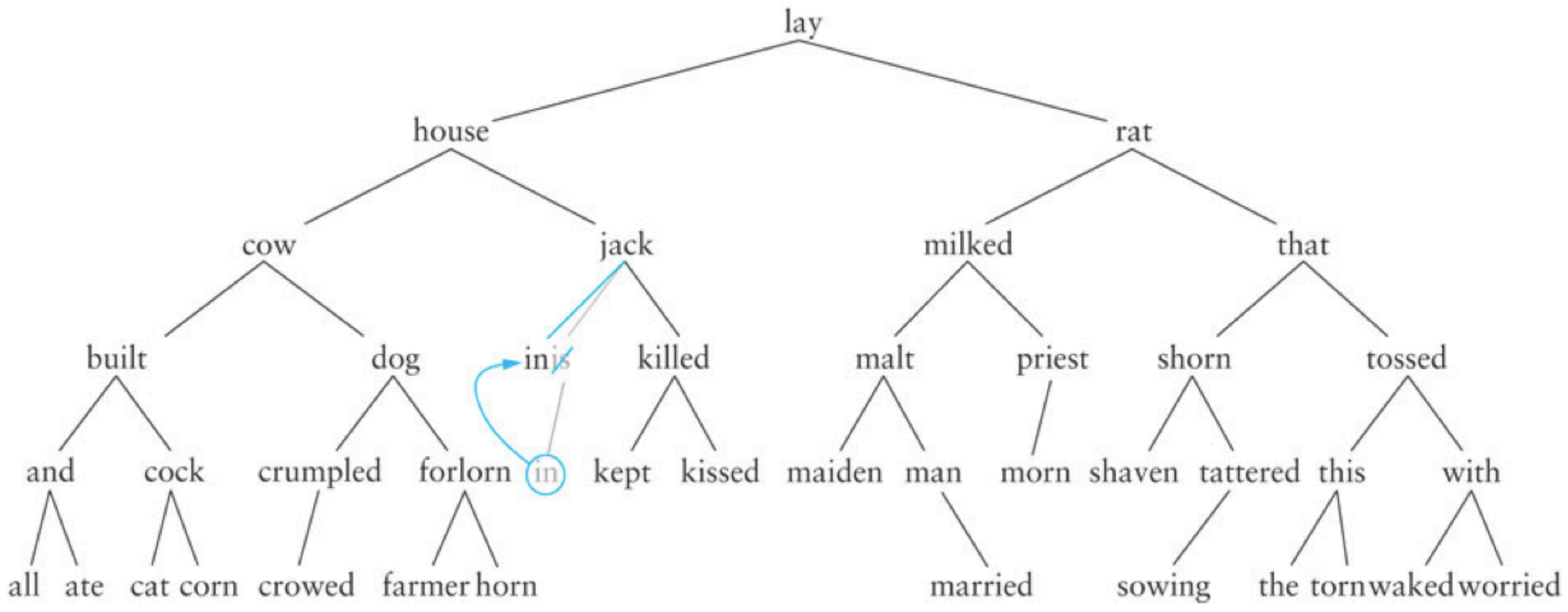
What is the order in which we should insert the values from a sorted array a[0]…a[N] into a BST in order to get a tree of minimal height ?

1. a[0], a[1]…a[N]
2. a[N/2], a[N-1],…,a[N/2+1], a[0]…,a[N/2-1]
3. a[N/2], a[N/4], a[3N/4],a[N/8],…
4. None of the above

govote.at
Code 954797

# Deleting from a BST

To delete a node with one child:
Deleting *is*, which has one child, *in* – we connect *in* to *is*'s parent *jack*



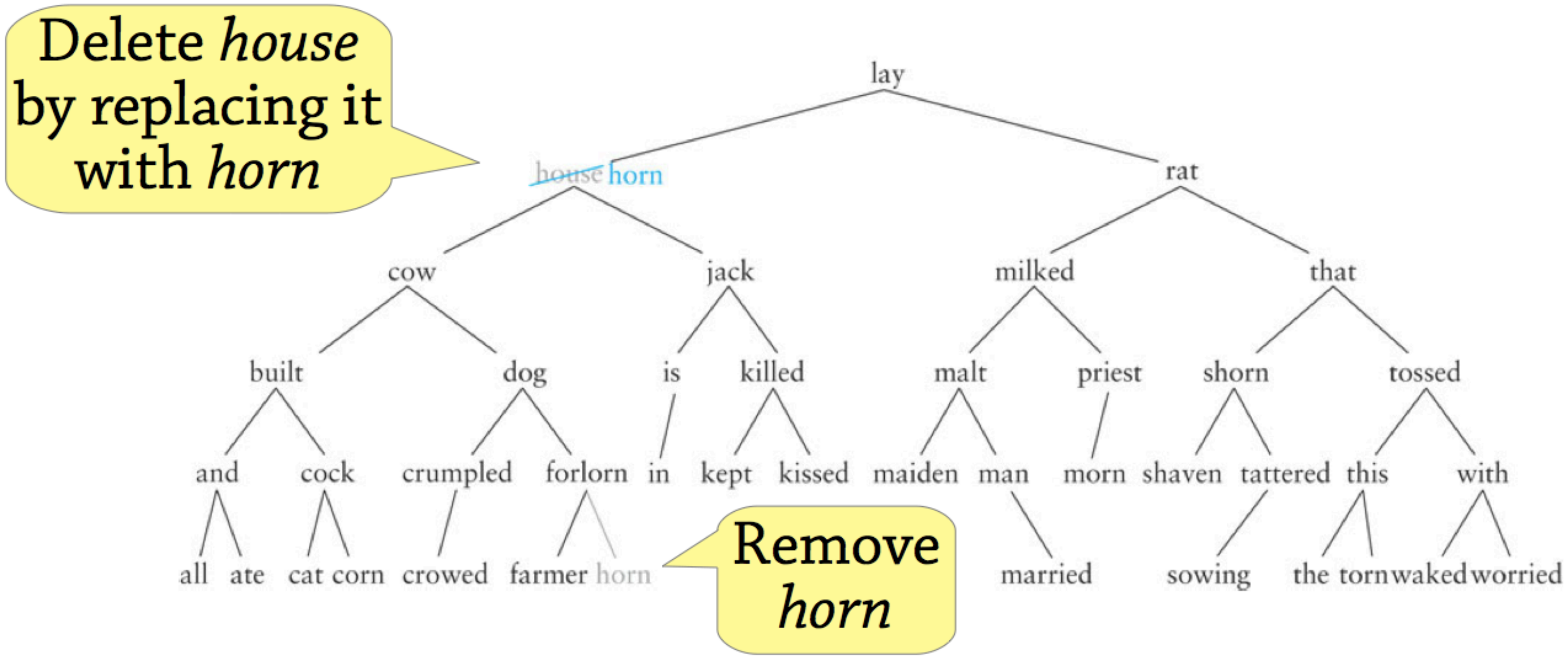
# Deleting from a BST

To delete a value from a BST:

- Find the node and its parent
- If it has no children, just remove it from the tree (by disconnecting it from its parent)
- If it has one child, replace the node with its child (by making the node's parent point at the child)
- If it has two children...?

# Deleting from a BST

Replace the deleted value with the biggest value from its left subtree (or the smallest from the right subtree)



## Question

Which of the following statements about the node from the left subtree that we replace the deleted node with is generally true ?

1. It can't have any children
2. It could have two children
3. It could have a left child
4. It could have a right child

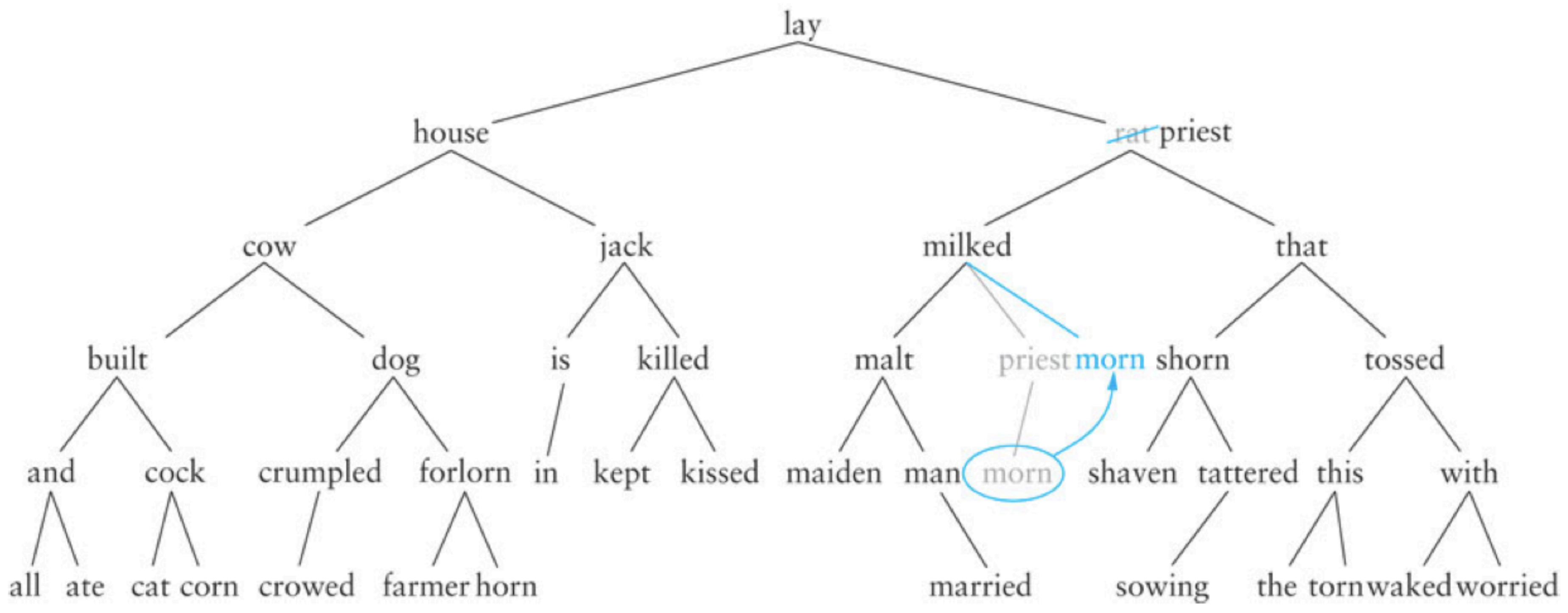
govote.at
Code ?

# Deleting from a BST

- Find the biggest value in the left subtree and put that value in the deleted node
- Using the biggest value preserves the invariant
- Biggest node = rightmost node
- Finally, delete the biggest value from the left subtree
- This node can't have two children (no right child), so deleting it is much easier

# Deleting from a BST

Deleting *rat*, we replace it with *priest*; now we have to delete *priest* which has a child, *morn*



# Complexity of BST Operations

All our operations are O(height of tree)
This means O(log n) if the tree is balanced, but O(n) if it's unbalanced.

Balanced BSTs add an extra invariant that makes sure the tree is balanced,
then all operations are O(log n)

Coming up soon!

# Summary of BSTs

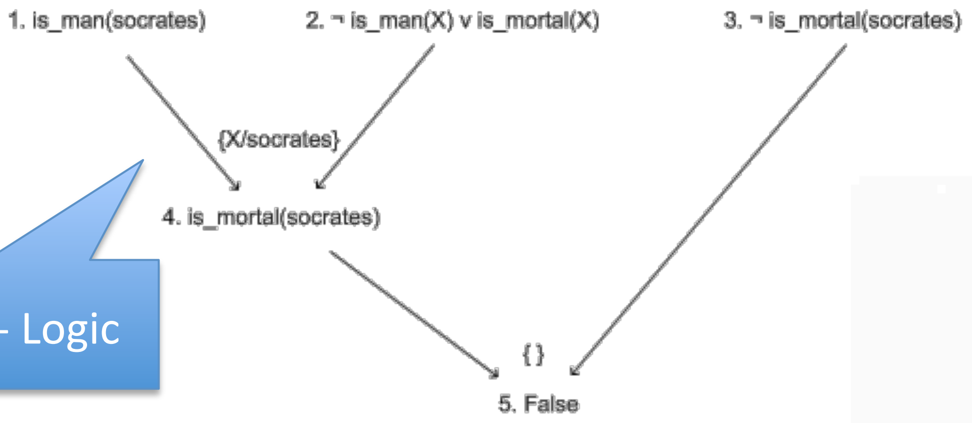
Can be used to implement sets and maps

- **lookup**: can easily find a value in the tree
- **insert**: perform a lookup, then put the new value at the place where the lookup would terminate
- **delete**: find the value, then several cases depending on how many children the node has

Complexity:

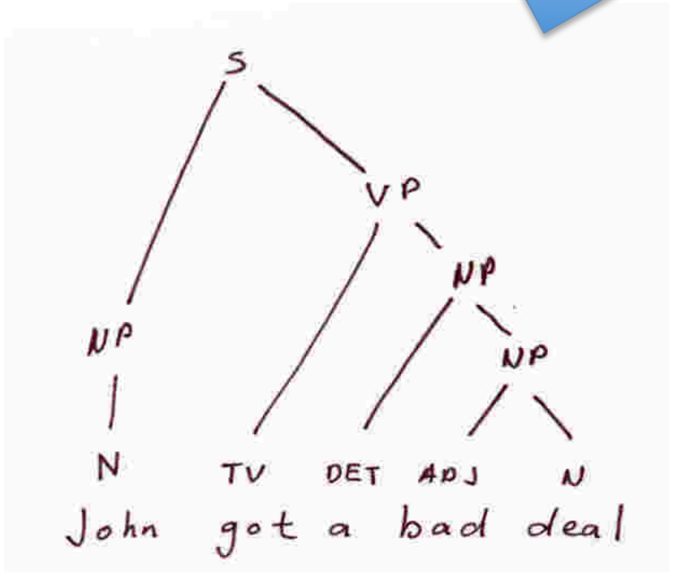
- all operations O(height of tree)
  that is, O(log n) if tree is balanced, O(n) if unbalanced
- inserting random data tends to give balanced trees, sequential data gives unbalanced ones

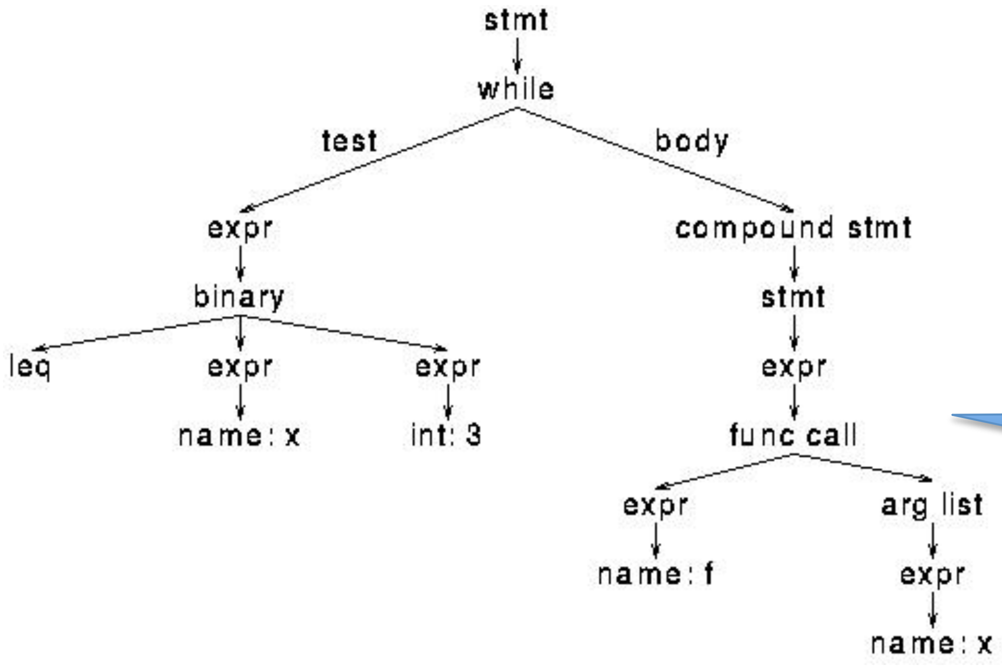
# More Trees in Computer Science



Proof tree - Logic

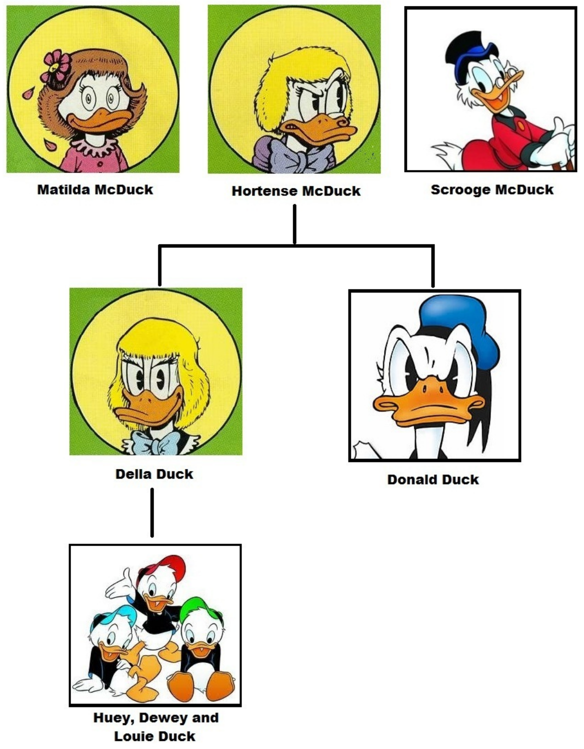
Parse trees – Natural Language Processing





Abstract Syntax Tress - Compilers

# To Do



Read from the book:
+ 4.1 – 4.3

Implement:
+ trees + the algorithms from today in your favourite programming language

Dugga next week:

http://www.cse.chalmers.se/edu/year/2014/course/DAT037_Datastrukturer/examination.html#dugga

Coming up:
+ advanced data structures
  - minimum spanning tree (trees and graphs)
  - balanced trees
  - more on sorting

# Questions about dugga