COMP250: Trees

Lecture 18

Jérôme Waldispühl

School of Computer Science

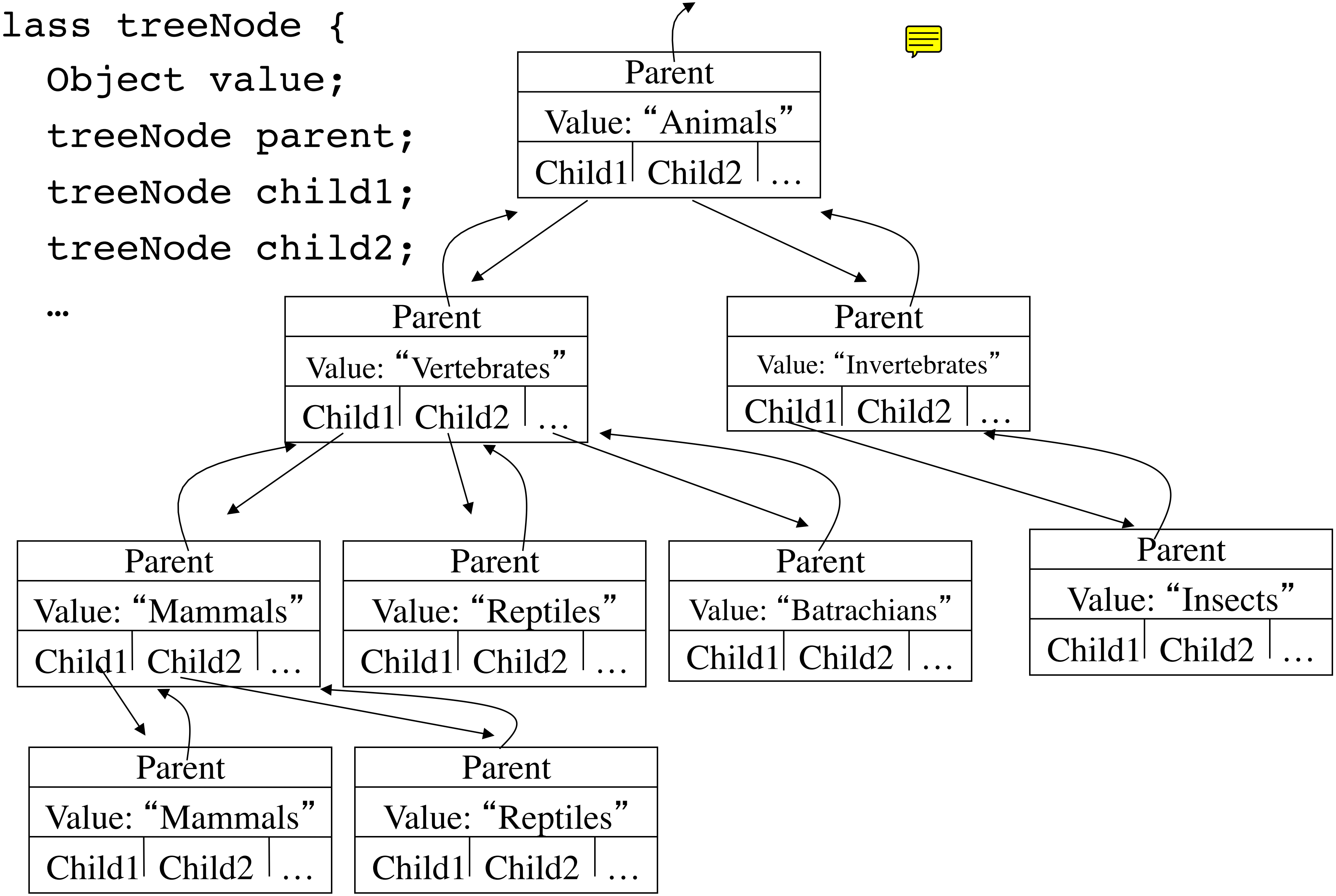
McGill University

Root

Leaves

Tree data structure

```
class treeNode {
   Object value;
   treeNode parent;
   treeNode child1;
   treeNode child2;
   …
}
```



Parent
Value: “Animals”
Child1 Child2 …

Parent
Value: “Vertebrates”
Child1 Child2 …

Parent
Value: “Invertebrates”
Child1 Child2 …

Parent
Value: “Mammals”
Child1 Child2 …

Parent
Value: “Reptiles”
Child1 Child2 …

Parent
Value: “Batrachians”
Child1 Child2 …

Parent
Value: “Insects”
Child1 Child2 …

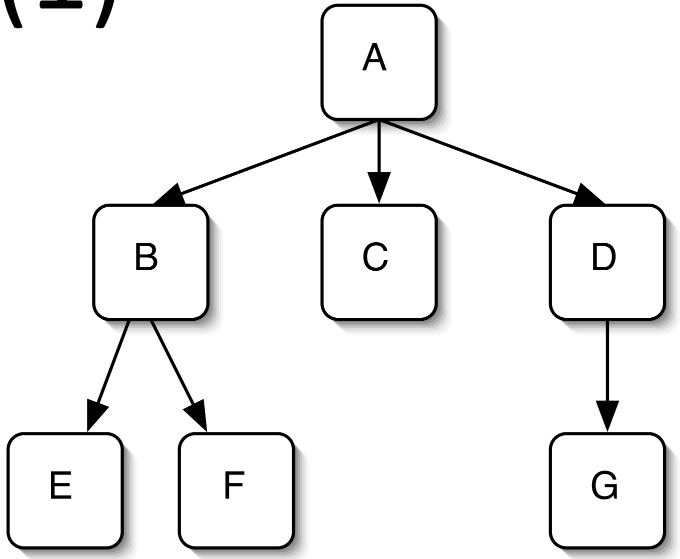
Parent
Value: “Mammals”
Child1 Child2 …

Parent
Value: “Reptiles”
Child1 Child2 …

Outline

- Terminology
- Examples of applications
- Exploring trees
- Implementing tree ADTs

Vocabulary (1)



Root: A (only node with parent==null)

Children(B) = E, F

Siblings(X) = {Nodes with the same parent as X, excluding X}

Siblings(B) = {C, D}, **Siblings**(A) = {}

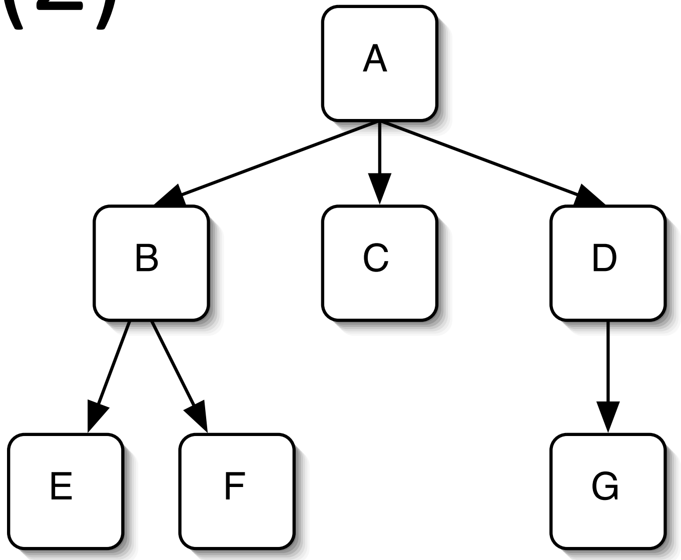
Descendants(X) = {Nodes below X}

Descendants(A) = {B,C,D,E,F,G}

Ancestors(X) = {Nodes between X and the root}

Ancestors(E) = {B, A}

Vocabulary (2)



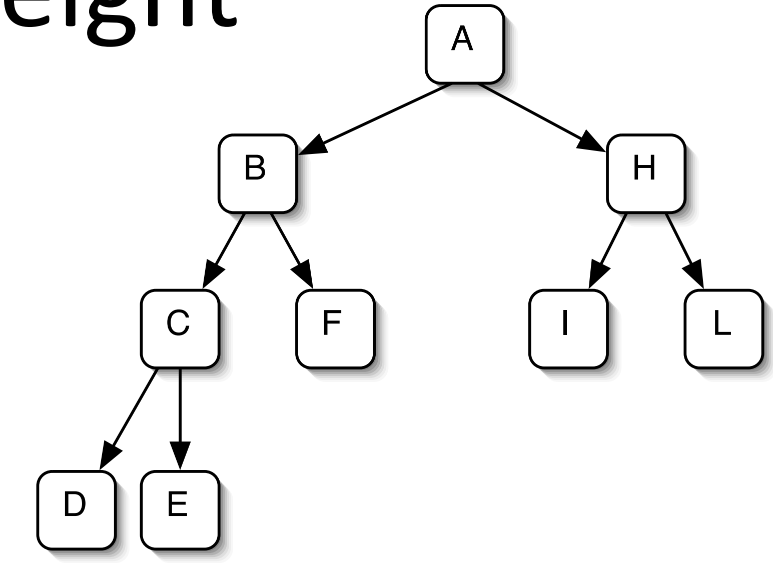
Nodes with no children are called **leaves**, or **external nodes**: {C, E, F, G}

Nodes with children are called **internal nodes**: {A, B, D}

A tree is **ordered** if the order of the children of a node matter

The **subtree rooted at X** is the tree of all descendents of X, including X.

Depth and Height



- **Depth of node x:**

  Depth(x) = number of ancestors of x

  Example: Depth(F) = 2

  Notice: Depth(x) = 1 + Depth(x.parent)

- **Height of a node x:**

  Height(x) = Number of nodes in the longest path between x and one of its descendant (excluding x)

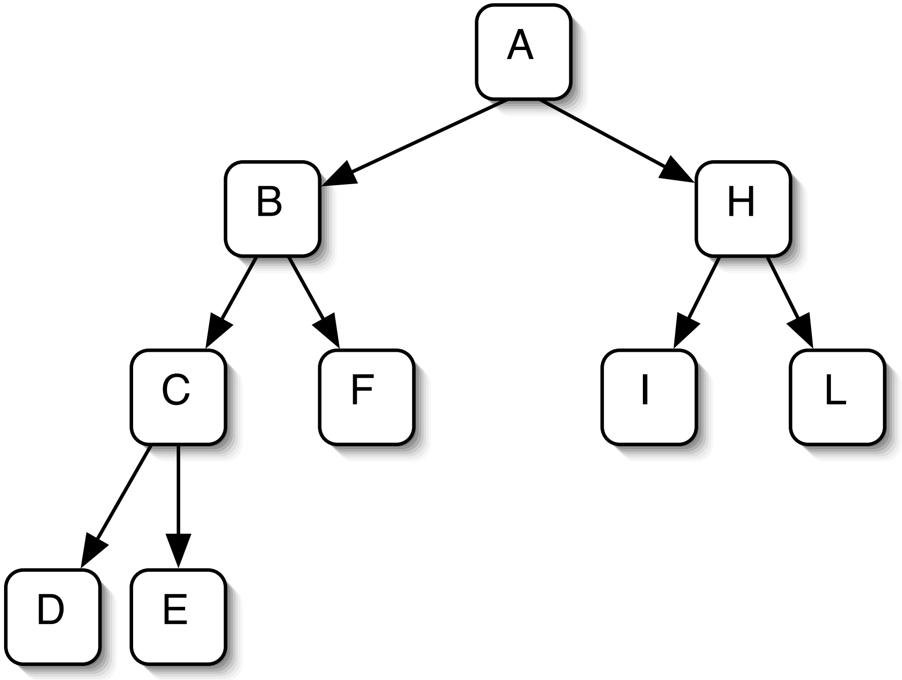
  Example: Height(B) = 2

  Notice:

  Height(x) = 1 + max(Height(x.leftChild), Height(x.rightChild));

- **Height of a tree = Height(root)**

Binary trees



- Each node has at most two children: left child and right child.
- Proper binary tree: each internal node has exactly two children.

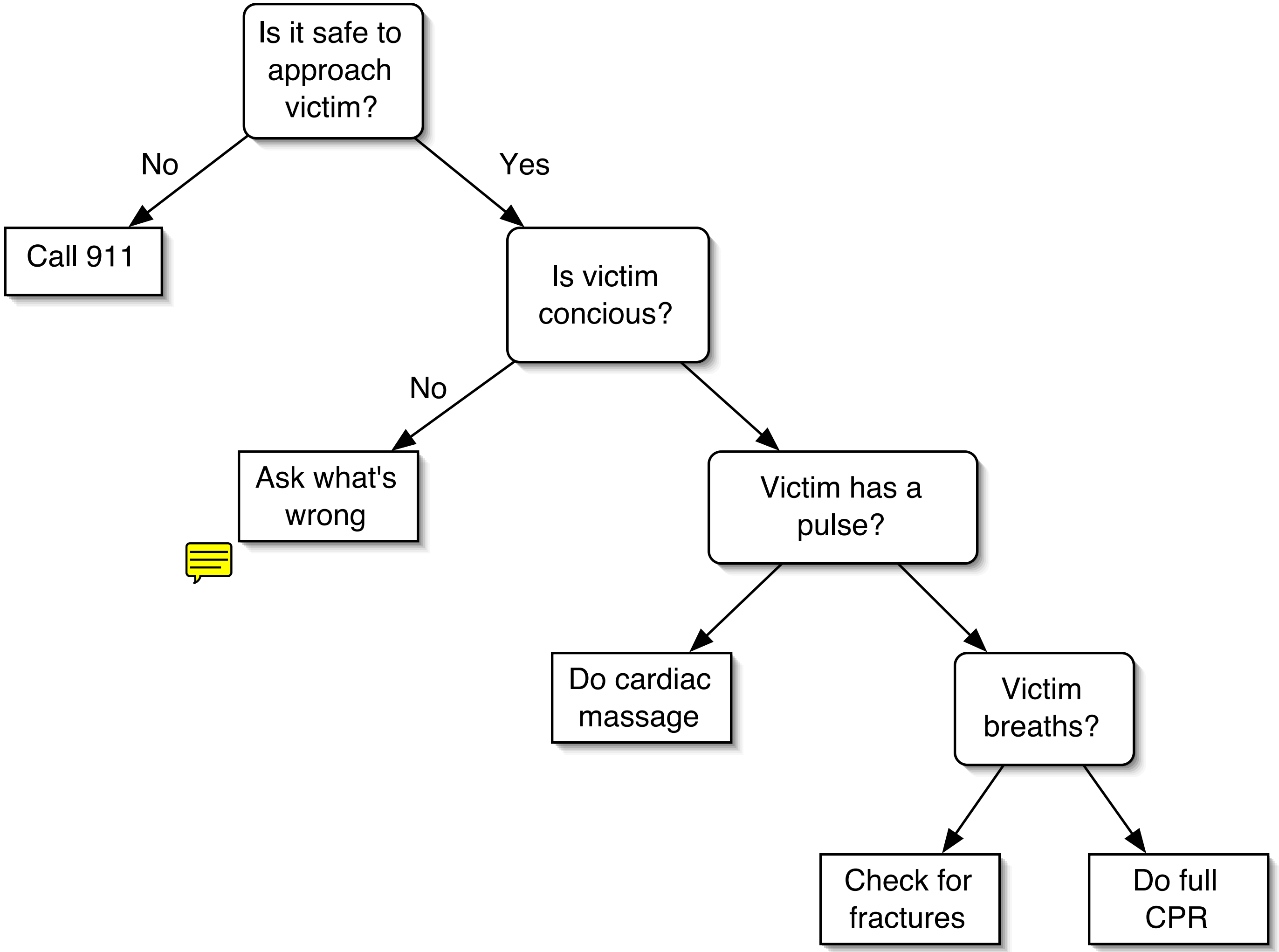
Applications

Trees in Computer Science

Many many applications:

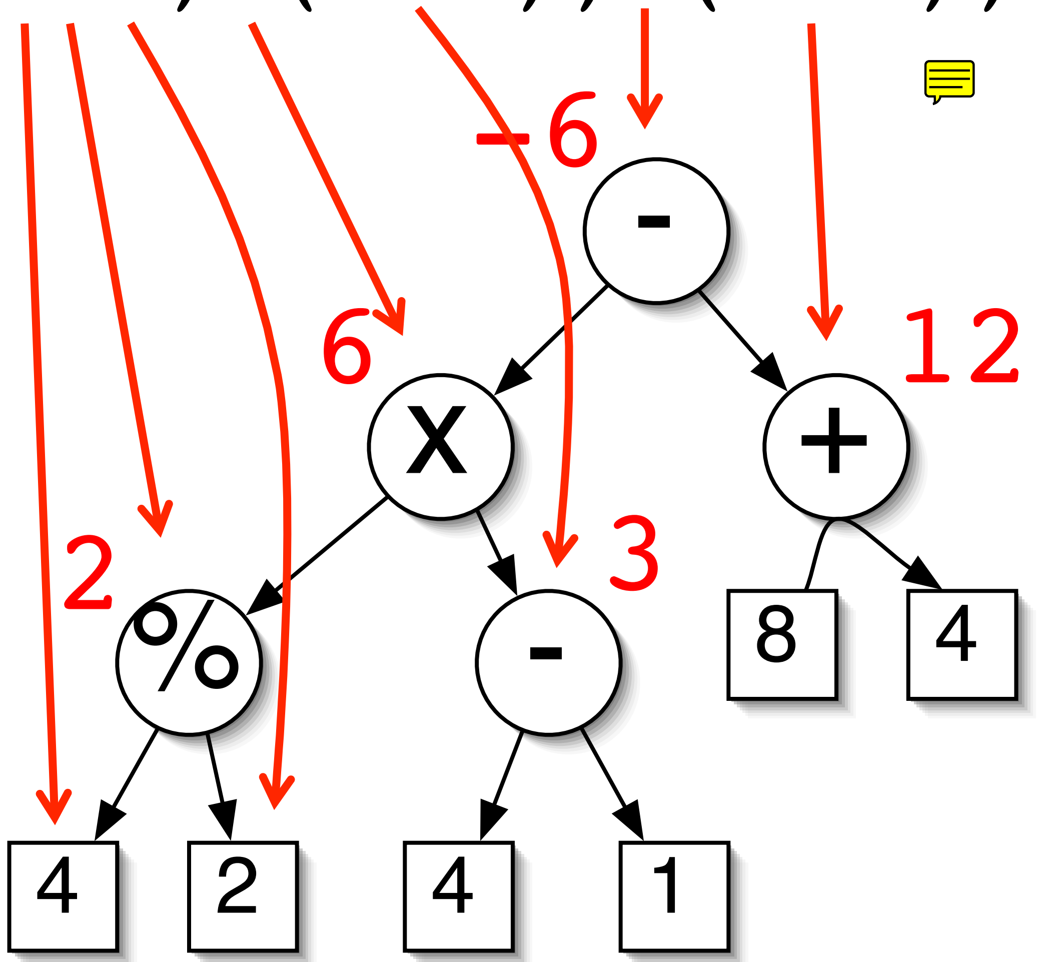
- Data storage
- Data compression
- Job scheduling
- Pattern matching
- Compilers
- Natural language processing
- Evolutionary biology (Phylogeny)

Decision trees



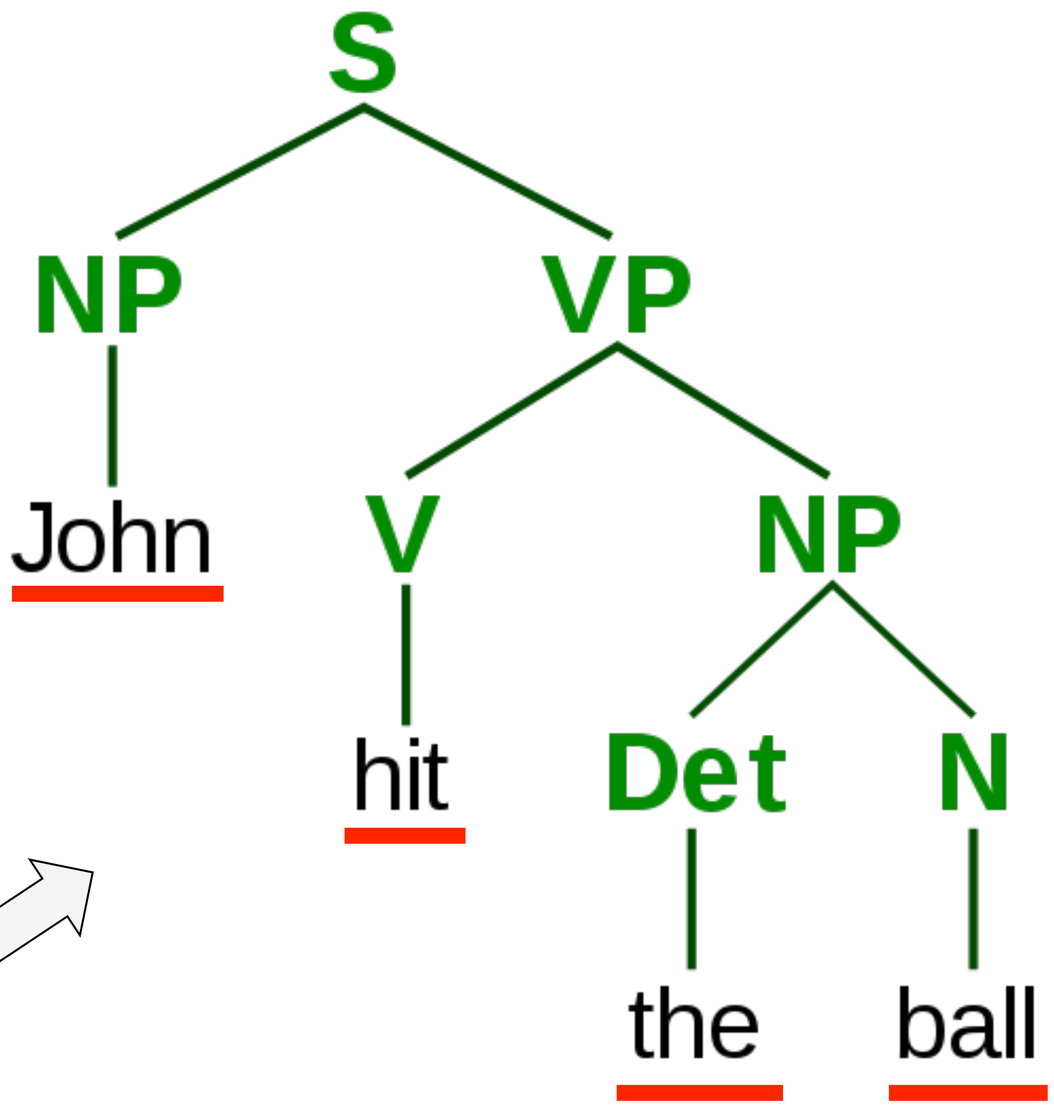
Representing and Evaluating Mathematical Expressions

```
((4%2)x(4-1))-(8+4))
```



Parse Tree

- S for sentence, the top-level structure.
- NP for noun phrase.
- VP for verb phrase.
- V for verb.
- D for determiner.
- N for noun.

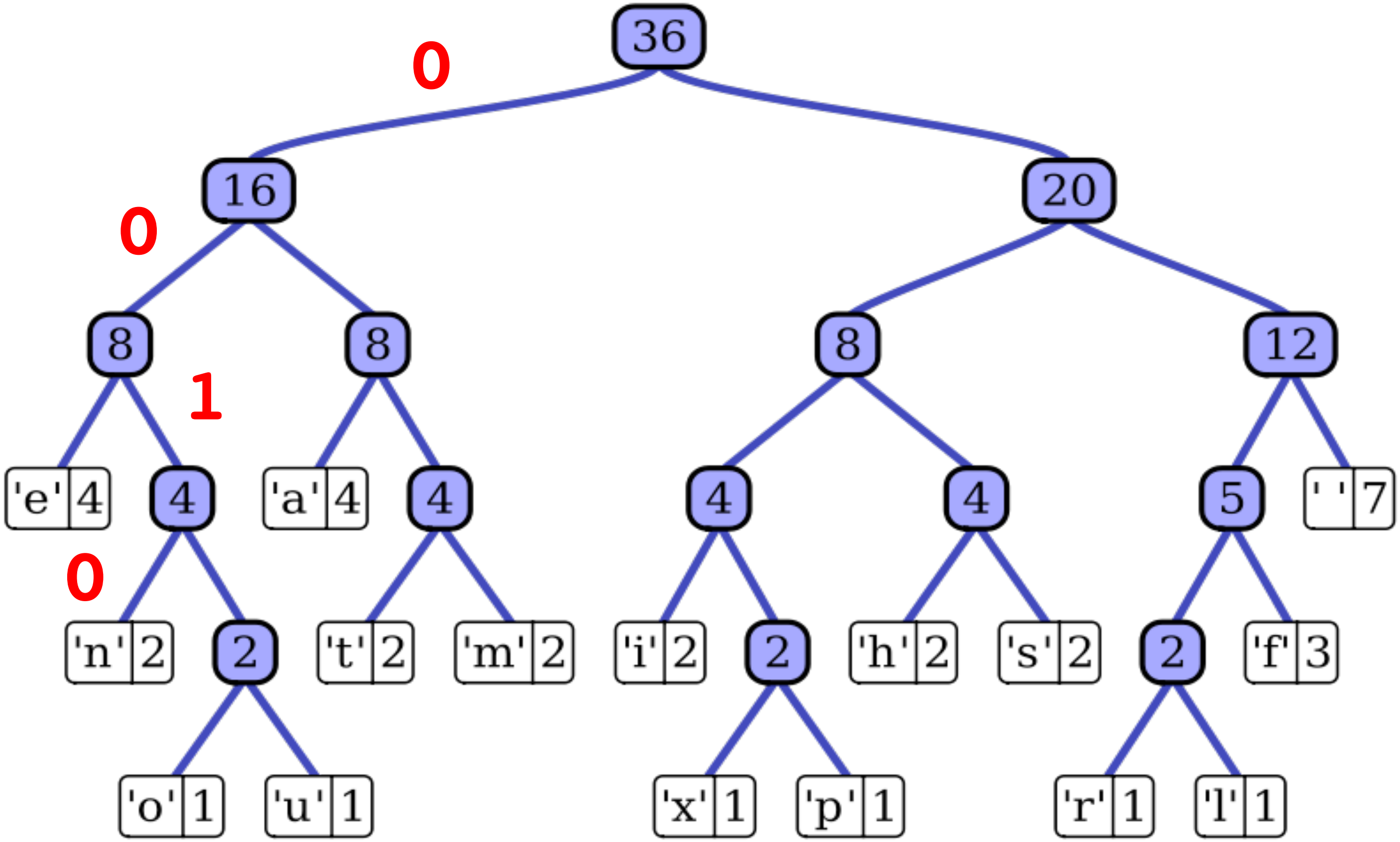


John hit the ball

http://en.wikipedia.org/wiki/Parse_tree

Huffman trees

Huffman tree encoding exact character frequencies of the text: "this is an example of a huffman tree".



http://en.wikipedia.org/wiki/Huffman_coding

'n' = 0010

Exploring trees

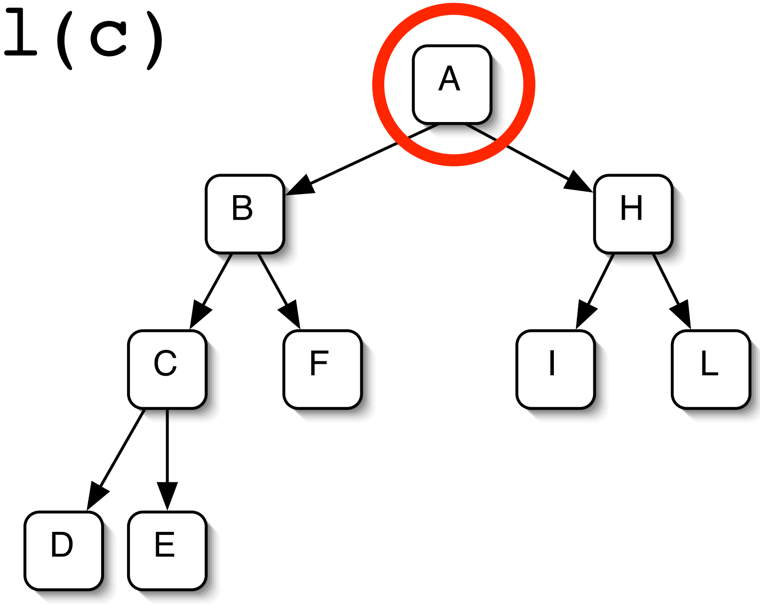
Traversing trees

- How to visit all nodes of a tree, starting from the root? Use recursion!!
- Pre-order traversal:

```
preorderTraversal(treeNode x)
   print x.value;
   for each c in children(x) do
         preorderTraversal(c)
```

- Output:

A B C D E F H I L



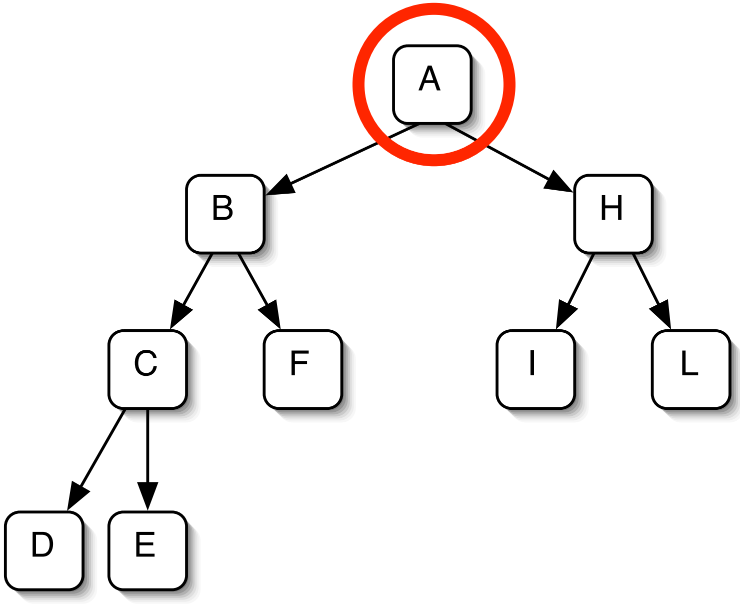
Traversing trees

- Post-order traversal:

```
postorderTraversal(treeNode x)
   for each c in children(x) do
         postorderTraversal(c);
   print x.value;
```

- Output:

D E C F B I L H A



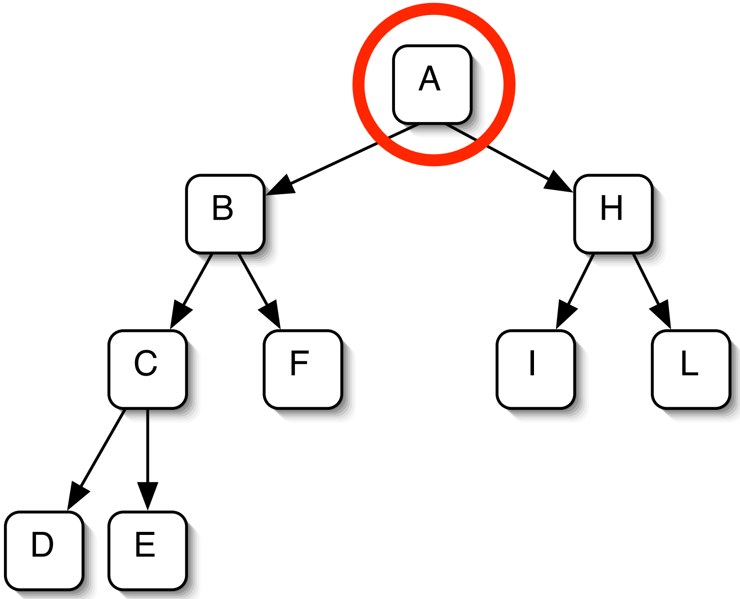
Traversing binary trees

- In-order traversal:

```
inorderTraversal(treeNode x)
   inorderTraversal(x.leftChild);
   print x.value;
   inorderTraversal(x.rightChild);
```

- Output:

D C E B F A I H L



Implementing trees

Binary tree ADT

Operations defined on a treeNode:

```
Object getValue();
treeNode getParent();
treeNode getLeftChild();
treeNode getRightChild();
treeNode getSibling();
void setParent(treeNode n)
void setLeftChild(treeNode n);
void setRightChild(treeNode n);
int depth(); // returns the depth of the node
int height(); // returns the height of the node
```

```
class treeNode {
   Object value;
   treeNode parent;
   treeNode left;
   treeNode right;
}

int depth() {
if (this.parent == null) { return 0; }
return 1 + depth(this.parent);
}

int height() {
if (this.left == null) { return 0; }
return 1 + Math.max(height(this.left),
                    height(this.right));
}
```