Root

Leaves
Tree data structure

class treeNode {
  Object value;
  treeNode parent;
  treeNode child1;
  treeNode 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}
Nodes without child 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** \( x \):
  \[ \text{Depth}(x) = \text{number of ancestors of } x \]
  Example: \( \text{Depth}(F) = 2 \)
  Notice: \( \text{Depth}(x) = 1 + \text{Depth}(x.\text{parent}) \)

• **Height of a node** \( x \):
  \[ \text{Height}(x) = \text{Number of nodes in the longest path between } x \]
  \[ \text{and one of its descendant (excluding } x) \]
  Example: \( \text{Height}(B) = 2 \)
  Notice:
  \[ \text{Height}(x) = 1 + \max(\text{Height}(x.\text{leftChild}), \text{Height}(x.\text{rightChild})) \]

• **Height of a tree** = \( \text{Height}(\text{root}) \)
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
Many many applications:

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

Trees in Computer Science
Victim has a pulse?

Victim breaths?

Is it safe to approach victim?

Is victim conscious?

Call 911

Ask what’s wrong

Victim has a pulse?

Victim breaths?

Check for fractures

Do full CPR

Do cardiac massage
Representing and Evaluating Mathematical Expressions

\[
( (4 \div 2) \times (4 - 1) ) - (8 + 4) 
\]
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:
  
  \[
  \text{postorderTraversal}(\text{treeNode } x) \\
  \quad \text{for each } c \text{ in children}(x) \text{ do} \\
  \quad \quad \text{postorderTraversal}(c); \\
  \quad \text{print } x.\text{value}; 
  \]

- Output:

```
D E C F B I L H A
```

```
Traversing binary trees

• In-order traversal:
  
  \[
  \text{inorderTraversal}(\text{treeNode } x) \\
  \text{inorderTraversal}(x.\text{leftChild}); \\
  \text{print } x.\text{value}; \\
  \text{inorderTraversal}(x.\text{rightChild}); \\
  \]

• 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));
}