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: “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
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 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** \textit{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** \textit{x}:
  
  \[ \text{Height}(x) = \text{number of nodes in the longest path between } x \]
  
  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** = \( \text{Height(root)} \)
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)
Victim has a pulse?

Victim breathes?

Decision trees

Is it safe to approach victim?

Call 911

Ask what's wrong

Is victim conscious?

Victim has a pulse?

Do cardiac massage

Check for fractures

Do full CPR
Representing and Evaluating Mathematical Expressions

\[((4 \% 2) \times (4 - 1)) - (8 + 4)\]
John hit the ball

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

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

'\text{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:

\[
\text{postorderTraversal}(\text{treeNode } x) \\
\text{for each } c \text{ in children}(x) \text{ do} \\
\text{postorderTraversal}(c); \\
\text{print } x.\text{value;}
\]

• Output:

\[
\text{D E C F B I L H A}
\]
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));
}