Trees

Make Money Fast!

- Stock Fraud
- Ponzi Scheme
- Bank Robbery
What is a Tree

- In computer science, a tree is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Applications:
  - Organization charts
  - File systems
  - Programming environments
Tree Terminology

- **Root**: node without parent (A)
- **Internal node**: node with at least one child (A, B, C, F)
- **External node (a.k.a. leaf)**: node without children (E, I, J, K, G, H, D)
- **Ancestors of a node**: parent, grandparent, grand-grandparent, etc.
- **Depth of a node**: number of ancestors
- **Height of a tree**: maximum depth of any node (3)
- **Descendant of a node**: child, grandchild, grand-grandchild, etc.

**Subtree**: tree consisting of a node and its descendants
Tree ADT

- We use positions to abstract nodes
- Generic methods:
  - integer size()
  - boolean isEmpty()
  - Iterator iterator()
  - Iterable positions()
- Accessor methods:
  - position root()
  - position parent(p)
  - Iterable children(p)
- Query methods:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- Update method:
  - element replace (p, o)
- Additional update methods may be defined by data structures implementing the Tree ADT
Preorder Traversal

- A traversal visits the nodes of a tree in a systematic manner.
- In a preorder traversal, a node is visited before its descendants.
- Application: print a structured document.

**Algorithm** \textit{preOrder}(v)

\textit{visit}(v)

for each child \( w \) of \( v \)

\textit{preorder}(w)
Postorder Traversal

- In a postorder traversal, a node is visited after its descendants
- Application: compute space used by files in a directory and its subdirectories

Algorithm \textit{postOrder}(v)

\begin{verbatim}
for each child w of v
    postOrder(w)
    visit(v)
\end{verbatim}
Binary Trees

- A binary tree is a tree with the following properties:
  - Each internal node has at most two children (exactly two for proper binary trees)
  - The children of a node are an ordered pair
- We call the children of an internal node left child and right child
- Alternative recursive definition: a binary tree is either
  - a tree consisting of a single node, or
  - a tree whose root has an ordered pair of children, each of which is a binary tree

Applications:
- arithmetic expressions
- decision processes
- searching
Arithmetic Expression Tree

- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands
- Example: arithmetic expression tree for the expression \((2 \times (a - 1) + (3 \times b))\)
Decision Tree

- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision

```
Want a fast meal?
  Yes
  How about coffee?
    Yes
    Starbucks
    No
    Spike’s
  No
  On expense account?
    Yes
    Al Forno
    No
    Café Paragon
```
Properties of Proper Binary Trees

- **Notation**
  - $n$: number of nodes
  - $e$: number of external nodes
  - $i$: number of internal nodes
  - $h$: height

- **Properties**: 
  - $e = i + 1$
  - $n = 2e - 1$
  - $h \leq i$
  - $h \leq (n - 1)/2$
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $h \geq \log_2 (n + 1) - 1$
BinaryTree ADT

- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Additional methods:
  - position left(p)
  - position right(p)
  - boolean hasLeft(p)
  - boolean hasRight(p)
- Update methods may be defined by data structures implementing the BinaryTree ADT
Inorder Traversal

- In an inorder traversal a node is visited after its left subtree and before its right subtree.
- Application: draw a binary tree.
  - $x(v) = \text{inorder rank of } v$
  - $y(v) = \text{depth of } v$

Algorithm \textit{inOrder}(v)

\begin{verbatim}
if hasLeft (v)
inOrder (left (v))
visit(v)
if hasRight (v)
inOrder (right (v))
\end{verbatim}
Print Arithmetic Expressions

- Specialization of an inorder traversal
  - print operand or operator when visiting node
  - print "(" before traversing left subtree
  - print ")" after traversing right subtree

Algorithm `printExpression(v)`

```plaintext
if hasLeft(v)
  print("(")
inOrder(left(v))
print(v.element())
if hasRight(v)
inOrder(right(v))
print (")")
```

```
((2 × (a - 1)) + (3 × b))
```
Evaluate Arithmetic Expressions

- Specialization of a postorder traversal
  - recursive method returning the value of a subtree
  - when visiting an internal node, combine the values of the subtrees

Algorithm `evalExpr(v)`

```plaintext
if isExternal(v)
    return v.element()
else
    x ← evalExpr(leftChild(v))
    y ← evalExpr(rightChild(v))
    ◊ ← operator stored at v
    return x ◊ y
```

© 2010 Goodrich, Tamassia
Euler Tour Traversal

- Generic traversal of a binary tree
- Includes a special cases the preorder, postorder and inorder traversals
- Walk around the tree and visit each node three times:
  - on the left (preorder)
  - from below (inorder)
  - on the right (postorder)
Linked Structure for Trees

- A node is represented by an object storing
  - Element
  - Parent node
  - Sequence of children nodes
- Node objects implement the Position ADT
Linked Structure for Binary Trees

- A node is represented by an object storing:
  - Element
  - Parent node
  - Left child node
  - Right child node

- Node objects implement the Position ADT
Array-Based Representation of Binary Trees

- Nodes are stored in an array $A$

  - Node $v$ is stored at $A[\text{rank}(v)]$
    - $\text{rank}(\text{root}) = 1$
    - If node is the left child of parent(node), $\text{rank}(\text{node}) = 2 \cdot \text{rank}(\text{parent}(\text{node}))$
    - If node is the right child of parent(node), $\text{rank}(\text{node}) = 2 \cdot \text{rank}(\text{parent}(\text{node})) + 1$
Template Method Pattern

- Generic algorithm
- Implemented by abstract Java class
- Visit methods redefined by subclasses
- Template method eulerTour
  - Recursively called on left and right children
  - A TourResult object with fields left, right and out keeps track of the output of the recursive calls to eulerTour

```java
public abstract class EulerTour<E, R> {
    protected BinaryTree<E> tree;
    public abstract R execute(BinaryTree<E> T);
    protected void init(BinaryTree<E> T) { tree = T; }
    protected R eulerTour(Position<E> v) {
        TourResult<R> r = new TourResult<R>();
        visitLeft(v, r);
        if (tree.hasLeft(p))
            { r.left=eulerTour(tree.left(v)); }
        visitBelow(v, r);
        if (tree.hasRight(p))
            { r.right=eulerTour(tree.right(v)); }
        return r.out;
    }
    protected void visitLeft(Position<E> v, TourResult<R> r) {}
    protected void visitBelow(Position<E> v, TourResult<R> r) {}
    protected void visitRight(Position<E> v, TourResult<R> r) {}
}
```
Specializations of EulerTour

- Specialization of class EulerTour to evaluate arithmetic expressions
- Assumptions
  - Nodes store ExpressionTerm objects with method getValue
  - ExpressionVariable objects at external nodes
  - ExpressionOperator objects at internal nodes with method setOperands(Integer, Integer)

```java
public class EvaluateExpressionTour extends EulerTour<ExpressionTerm, Integer> {
    public Integer execute(BinaryTree<ExpressionTerm> T) {
        init(T);
        return eulerTour(tree.root());
    }

    protected void visitRight(Position<ExpressionTerm> v, TourResult<Integer> r) {
        ExpressionTerm term = v.element();
        if (tree.isInternal(v)) {
            ExpressionOperator op = (ExpressionOperator) term;
            op.setOperands(r.left, r.right);
        }
        r.out = term.getValue();
    }
}
```

© 2010 Goodrich, Tamassia