Trees

Data structures and Algorithms

Acknowledgement:
These slides are adapted from slides provided with *Data Structures and Algorithms in C++* by Goodrich, Tamassia and Mount (Wiley, 2004)
Outline and Reading

- Tree ADT (§7.1.2)
- Preorder and postorder traversals (§7.2)
- BinaryTree ADT (§7.3)
- Inorder traversal (§7.3.6)
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
Tree Terminology

- **Root**: node without parent (A)
- **Internal node**: node with at least one child (A, B, C, F)
- **Leaf** (aka **External node**): node without children (E, I, J, K, G, H, D)
- **Ancestors** of a node: parent, grandparent, great-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, great-grandchild, etc.

**Subtree**: tree consisting of a node and its descendants
Exercise: Trees

Answer the following questions about the tree shown on the right:

- What is the size of the tree (number of nodes)?
- Classify each node of the tree as a root, leaf, or internal node.
- List the ancestors of nodes B, F, G, and A. Which are the parents?
- List the descendents of nodes B, F, G, and A. Which are the children?
- List the depths of nodes B, F, G, and A.
- What is the height of the tree?
- Draw the subtrees that are rooted at node F and at node K.
Tree ADT

We use positions to abstract nodes

Generic methods:
- integer size()
- boolean isEmpty()
- objectIterator elements()
- positionIterator positions()

Accessor methods:
- position root()
- position parent(p)
- positionIterator children(p)

Query methods:
- boolean isInternal(p)
- boolean isLeaf (p)
- boolean isRoot(p)

Update methods:
- swapElements(p, q)
- object replaceElement(p, o)

Additional update methods may be defined by data structures implementing the Tree ADT
Depth and Height

$\nu$: a node of a tree $T$.

- The **depth** of $\nu$ is the number of ancestors of $\nu$, excluding $\nu$ itself.

- The **height** of a node $\nu$ in a tree $T$ is defined recursively:
  - If $\nu$ is an external node, then the height of $\nu$ is 0.
  - Otherwise, the height of $\nu$ is one plus the maximum height of a child of $\nu$.

**Algorithm** $depth(T, \nu)$

```
if $T.isRoot(\nu)$
  return 0
else
  return 1 + $depth(T, T.parent(\nu))$
```

**Algorithm** $height(T, \nu)$

```
if $T.isExternal(\nu)$
  return 0
else
  $h \leftarrow 0$
  for each child $w$ of $\nu$ in $T$
    $h \leftarrow \max(h, height(T, w))$
  return 1 + $h$
```
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 $\text{preOrder}(v)$

$\text{visit}(v)$

for each child $w$ of $v$

$\text{preOrder}(w)$
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 $postOrder(v)$
- for each child $w$ of $v$
  - $postOrder(w)$
  - $visit(v)$

Trees
Binary Tree

- A binary tree is a tree with the following properties:
  - Each internal node has two children
  - 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
  - leaves: 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
  - leaves: decisions
- Example: shooting (robots playing football)

```
See the ball?

Yes
Ball and goal in line?
Yes Forwards
No Adjust position

No
Ball last seen on the left?
Yes Turn left
No Turn right
```
Properties of Binary Trees

• Notation
  \( n \) number of nodes
  \( l \) number of leaves
  \( i \) number of internal nodes
  \( h \) height

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

- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Update methods may be defined by data structures implementing the BinaryTree ADT
- Additional methods:
  - position `leftChild(p)`
  - position `rightChild(p)`
  - position `sibling(p)`
Inorder Traversal

- In an *inorder traversal*, a node is visited after its left subtree and before its right subtree

**Algorithm** `inOrder(v)`

```
if isInternal(v)
    inOrder(leftChild(v))
visit(v)
if isInternal(v)
inOrder(rightChild(v))
```

Trees
Inorder Traversal – Application

- Application: draw a binary tree. Assign $x$- and $y$-coordinates to node $v$, where
  - $x(v) = \text{inorder rank of } v$
  - $y(v) = \text{depth of } v$
Exercise: Preorder & InOrder Traversal

- Draw a (single) binary tree $T$, such that
  - Each internal node of $T$ stores a single character
  - A preorder traversal of $T$ yields EXAMFUN
  - An inorder traversal of $T$ yields MAFXUEN
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 \textit{printExpression}(v)
\begin{verbatim}
if hasLeft(v)
    print ("(")
    printExpression(leftChild(v))
print(v.element())
if hasRight(v)
    printExpression(rightChild(v))
    print (")")
\end{verbatim}

\[(2 \times (a - 1)) + (3 \times b)\]
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 \textit{evalExpr}(v)

\begin{center}
\begin{aligned}
\text{if } & \text{isExternal}(v) \\
& \quad \text{return } v.\text{element}() \\
\text{else} & \\
& \quad x \leftarrow \text{evalExpr(leftChild}(v)) \\
& \quad y \leftarrow \text{evalExpr(rightChild}(v)) \\
& \quad \bigcirc \leftarrow \text{operator stored at } v \\
& \quad \text{return } x \bigcirc y
\end{aligned}
\end{center}
Exercise: Arithmetic Expressions

• Draw an expression tree that has
  • Four leaves, storing the values 1, 5, 6, and 7
  • 3 internal nodes, storing operations +, -, *, /
    (operators can be used more than once, but each
    internal node stores only one)
  • The value of the root is 21
Data Structure for Trees

- A node is represented by an object storing:
  - Element
  - Parent node
  - Sequence of children nodes
- Node objects implement the Position ADT
Data 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
C++ Implementation

- Tree interface
- BinaryTree interface extending Tree
- Classes implementing Tree and BinaryTree and providing
  - Constructors
  - Update methods
  - Print methods
- Examples of updates for binary trees
  - expandExternal(v)
  - removeAboveExternal(w)