Outline and Reading

- Tree ADT (§6.1)
- Preorder and postorder traversals (§6.2.3)
- BinaryTree ADT (§6.3.1)
- Inorder traversal (§6.3.4)
- Euler Tour traversal (§6.3.4)
- Template method pattern (§6.3.5)
- Data structures for trees (§6.4)
- C++ implementation (§6.4.2)

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.

Tree ADT

We use positions to abstract nodes.

Generic methods:
- integer size()
- boolean isEmpty()
- object Iterator elements()
- position Iterator positions()

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

Query methods:
- boolean isInternal(p)
- boolean isExternal(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.

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.

Algorithm preorder(v)
for each child w of v
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)

Example:

- homeworks/ 3K
- h1c.doc 2K
- h1n.doc 10K
- DDR.java 25K
- Stocks.java 20K
- Robot.java 20K
- todo.txt 1K

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
  - 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 No
  - How about coffee? Yes No
  - On expense account? Yes No
  - Starbucks Spike's Al Forno Café Paragon

Properties of Binary Trees

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

- Properties:
  - $e = i + 1$
  - $n = 2e - 1$
  - $i \leq h$
  - $h \leq \frac{n - 1}{2}$
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $h \geq \log_2 (n + 1) - 1$

BinaryTree ADT

- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT.
- Additional methods:
  - position leftChild(p)
  - position rightChild(p)
  - position sibling(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{itemize}
  \item if isInternal(v)
    \begin{itemize}
      \item \textit{inOrder}(\text{leftChild}(v))
      \item visit(v)
      \item \textit{inOrder}(\text{rightChild}(v))
    \end{itemize}
\end{itemize}

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{itemize}
  \item if isInternal(v)
    \begin{itemize}
      \item print("(")
      \item \textit{inOrder}(\text{leftChild}(v))
      \item print(v.element())
      \item \textit{inOrder}(\text{rightChild}(v))
      \item print(")")
    \end{itemize}
\end{itemize}

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{itemize}
  \item if isExternal(v)
    \begin{itemize}
      \item return v.element()
    \end{itemize}
  \item else
    \begin{itemize}
      \item x ← \textit{evalExpr}(\text{leftChild}(v))
      \item y ← \textit{evalExpr}(\text{rightChild}(v))
      \item ◊ ← \text{operator stored at v}
      \item return \(x \text{ ◊ } y\)
    \end{itemize}
\end{itemize}

Euler Tour Traversal

- Generic traversal of a binary tree.
  Includes a special case 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)

\[
((2 \times (a - 1)) + (3 \times b))
\]

Template Method Pattern

- Generic algorithm that can be specialized by redefining certain steps.
- Implemented by means of an abstract C++ class.
- Visit methods that can be redefined by subclasses.

\begin{verbatim}
class EulerTour {
  protected:
    BinaryTree* tree;
  virtual void visitExternal(Position p, Result r) { }
  virtual void visitLeft(Position p, Result r) { }
  virtual void visitBelow(Position p, Result r) { }
  virtual void visitRight(Position p, Result r) { }
  int eulerTour(Position p) {
    Result r = initResult();
    if (tree->isExternal(p)) { visitExternal(p, r); }
    else {
      visitLeft(p, r);
      r.leftResult = eulerTour(tree->leftChild(p));
      visitBelow(p, r);
      rightResult = eulerTour(tree->rightChild(p));
      visitRight(p, r);
      return r.finalResult;
    } // … other details omitted
  }
}
\end{verbatim}

Specializations of EulerTour

- We show how to specialize class EulerTour to evaluate an arithmetic expression.

Assumptions
- External nodes support a function \text{value}(), which returns the value of this node.
- Internal nodes provide a function \text{operation}(int, int), which returns the result of some binary operator on integers.

\begin{verbatim}
class EvaluateExpression : public EulerTour {
  protected:
    virtual void visitExternal(Position p, Result r) {
      r.finalResult = p.element().value();
    }
    virtual void visitRight(Position p, Result r) {
      Operator op = p.element().operator();
      r.finalResult = p.element().operation(r.leftResult, r.rightResult);
    }
    // … other details omitted
}
\end{verbatim}
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)