Binary Trees (2)

Outline and Required Reading:

- Binary Trees (§ 6.3)
- Data Structures for Representing Trees (§ 6.4)
Traversal of Binary Tree: Preorder

**Preorder Traversal** — parent is processed previous to its children (i.e. left and right subtree)

- start at the root and recursively perform:
  1) process node;
  2) do preorder traversal on left child;
  3) do preorder traversal on right child.

**Implementation**

```java
public void binaryPreorder(Tree T, Position v) {
    Object e = v.element();
    process(e);                  /* do something with v */
    if  (T.isInternal(v)) { /* if v is internal, “visit” children */
        binaryPreorder(T, T.leftChild(T,v));
        binaryPreorder(T, T.rightChild(T,v)); }
}
```
Traversals of Binary Tree: Postorder

**Postorder Traversal** — processing of parent-node is postponed until both children-nodes are processed

- start at the root and recursively perform:
  1) do postorder traversal on left subtree;
  2) do postorder traversal on right subtree;
  3) process node.

**Implementation**

```java
public void binaryPostorder(Tree T, Position v) {
    if (T.isInternal(v)) { /* if v is internal, “visit” children */
        binaryPostorder(T, T.leftChild(T,v));
        binaryPostorder(T, T.rightChild(T,v)); 
    }
    Object e = v.element();
    process(e); /* do something with v */
}
```

Note: the first processed node is the leftmost descendant of the root!
Traversals of Binary Tree: Postorder

Example 1  [ postorder traversal ]

Write the sequence that would be generated by printing the content of each node during a postorder traversal of the given tree.
Traversals of Binary Tree: Inorder

**Inorder Traversal** — parent node is processed *in* between “visits” to its children \(\Rightarrow\) tree is “visited” from left to right

- start at the root and recursively perform:
  1) do inorder traversal on left subtree;
  2) process node;
  3) do inorder traversal on right subtree.

**Implementation**

```java
public void binaryInorder(Tree T, Position v) {
  if (T.isInternal(v)) {
    binaryInorder(T, T.leftChild(T, v));
  }
  Object e = v.element();
  process(e);
  if (T.isInternal(v)) {
    binaryInorder(T, T.rightChild(T, v));
  }
}
```
Traversal of Binary Tree (cont.)

Example 2  [ preorder, postorder, inorder traversal ]

Preorder Traversal  Inorder Traversal  Postorder Traversal
60  20  10  40  30  50  70  10  20  30  40  50  60  70  10  30  50  40  20  70  60
Example 3  [ depth of nodes ]

Give an $O(n)$-time algorithm for computing the depth of all nodes in a tree $T$, where $n$ is the number of nodes of $T$.

\[
\text{depth}_{\text{child}} = \text{depth}_{\text{parent}} + 1
\]

Assume each node has a field (variable) depth, where its actual depth is stored. When doing the visit in “preorder traversal”, store the depth of the nodes’ parent incremented by 1.

```java
public void binaryPreorder(Tree T, Position v) {
    if ( v == isRoot() ) v.setDepth(0);
    else v.setDepth( ((BTNode) v.getParent()).getDepth() + 1 );
    if (T.isInternal(v)) {
        binaryPreorder(T, T.leftChild(T,v));
        binaryPreorder(T, T.rightChild(T,v)); }
}
```
Example 4 [ evaluation of arithmetic expressions ]

Conventional infix notation – places the operator between two values that it operates on.

“Inorder traversal with parenthesis” produces infix expression.

```
public void binaryInorder(Tree T, Position v) {
    if (T.isInternal(v)) {
        System.out.println("(");
        binaryInorder(T, T.leftChild(T,v));
        Object e = v.element();
        System.out.println(e);
        if (T.isInternal(v)) {
            binaryInorder(T, T.rightChild(T,v));
            System.out.println(")");
        }
    }
}
```

Traversal of Binary Tree (cont.)
Problems with infix notation – we need to keep parenthesis around each internal expression.

In prefix and postfix notation parenthesis are not necessary!

\begin{align*}
\text{Inorder traversal} & \Rightarrow \text{infix expression:} & a \times (b - c) \\
\text{Preorder traversal} & \Rightarrow \text{prefix expression:} & x \ a - b \ c \\
\text{Postorder traversal} & \Rightarrow \text{postfix expression:} & a \ b \ c - x
\end{align*}

What is the advantage of storing an arithmetic expression in a binary tree???
Algorithm for evaluation of postfix expressions:

declare a local Stack variable [e.g. S];
current = first element in the expression;
while current is not null {
    if current is an operand, push it on S;
    else
        pop two operands from S [e.g. E₁, E₂];
        push (E₁ current E₂) back on S;
    set current to the next element in the expression;
}

inorder traversal / infix notation: a x ( b – c )
postorder traversal / postfix notation: a b c - x

For exercise, implement this algorithm in Java!
Algorithm for evaluation of expressions stored in binary trees:

variable currentNode is passed to this method;
if currentNode is external, return currentNode.value;
else
  E₁ = call this method on leftChild;
  E₂ = call this method on rightChild;
  return (E₁ currentNode.value E₂)

For exercise, implement this algorithm in Java!
**Traversals of Binary Trees: Euler Tour**

**Euler Tour Traversal** — generic traversal — “walk around” the tree and process each node 3 times: on left, from below on right

- preorder, inorder, postorder traversals are special cases of the Euler tour traversal

```java
public void eulerTour(Tree T, Position v) {
    Object e = v.element(); ... /* do something with v */
    if (T.isInternal(v)) {
        eulerTour(T, T.leftChild(T,v));
    }
    Object e = v.element(); ... /* do something with v */
    if (T.isInternal(v)) {
        eulerTour(T, T.rightChild(T,v));
    }
    Object e = v.element(); ... /* do something with v */
}
```
Saving and Restoring Binary Trees

Why and How to Save Binary Trees?

Imagine a program that creates a large database in a form of a binary tree;

- the content of the tree must be stored in a file if we want to recover and use it at a later time
- the stored information must enable reconstruction of an exact copy of the original tree

Saving Trees Using Traversals – results of a single traversal cannot enable perfect reconstruction of the original binary tree

Example: “equivalent” trees with respect to preorder traversal (A, E, D, B, C, F)
Example 5  [ save/reconstruct using preorder traversal + additional info ]

Steps during “Save Tree” procedure:

(1) walk the tree in preorder

(2) at each node, write (O,D) pair:
    O - “content” of the node
    D - describes whether the node has left, right, both or neither subtrees

(A,2), (E,0), (D,2), (B,0), (C,R), (F,0)

(A,2), (E,2), (D,0), (B,0), (C,L), (F,0)
Reconstruction through Recursion:

```java
public Tree rebuildTree() {
    /* read next (O,D) pair from the file */
    t = new Tree(O);    /* create new tree and place O in the root */
    if  ( (D==2) || (D==L) ) {
        t.attachL(rebuildTree()); };   /* create new subtree and attach it to the left - try to implement ! */
    if  ( (D==2) || (D==R) ) {
        t.attachR(rebuildTree()); };   /* create new subtree and attach it to the right - try to implement ! */
    return t;
}
```
Saving and Restoring Binary Trees  (cont.)

(A, 2), (E, 0), (D, 2), (B, 0), (C, R), (F, 0)

(A, 2), (E, 0), (D, 2), (B, 0), (C, R), (F, 0)

(A, 2), (E, 0), (D, 2), (B, 0), (C, R), (F, 0)
(A,2), (E,0), (D,2), (B,0), (C,R), (F,0)

(A,2), (E,0), (D,2), (B,0), (C,R), (F,0)

etc.
Representing General Trees with Binary Trees

**Transformation Procedure** for transforming a general ordered tree (T) into a binary tree (T’):

1. the root of the original tree is the root of the binary tree
2. if u is an internal node of the original tree and v is the 1st child of u, then v will be the left child of u in the binary tree
3. if v has a sibling w in the original tree which immediately follows it, then w will be the right child of v in the binary tree

Set of siblings in T = chain of right children in T’ rooted at the 1st sibling!
Representing General Trees with Binary Trees (cont.)

Binary Tree ADT is sufficient to represent any tree structure.

NOTE:

(1) preorder traversal of the binary tree representation corresponds to preorder traversal of the original tree

(2) inorder traversal of the binary tree representation corresponds to postorder traversal of the original tree
Representing General Trees with Binary Trees (cont.)

Implementation of General Trees as Binary Trees - Performance

– resulting binary tree is \textit{unbalanced} \Rightarrow “linked structure” implementation is optimal

– all methods, except \texttt{positions()}, \texttt{elements()}, \texttt{children()}, \texttt{parent()} run in $O(1)$ time

<table>
<thead>
<tr>
<th>Method</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{size}, \texttt{isEmpty}</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>\texttt{positions}, \texttt{elements}</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>\texttt{swapElements}, \texttt{replaceElement}</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>\texttt{leftChild}, \texttt{rightChild}, \texttt{sibling}</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>\texttt{root}</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>\texttt{parent}(v)</td>
<td>$O(s_v)$</td>
</tr>
<tr>
<td>\texttt{children}(v)</td>
<td>$O(c_v)$</td>
</tr>
<tr>
<td>\texttt{isInternal}, \texttt{isExternal}, \texttt{isRoot}</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

$s_v$ - number of siblings of $v$!
Q.1 Suppose we have the following recurrence relation for $f(n)$:

$$
\begin{align*}
    f(0) &= 0 & \text{for } n=0; \\
    f(n) &= f(n-1) + n & \text{for } n>0;
\end{align*}
$$

Prove by induction on $n$ that the following is a closed form formula for $f$:

$$f(n) = \frac{n(n+1)}{2}$$

Q.2 Let $BT$ be a complete binary tree with $n$ nodes. Let $h(n)$ denoted the height of this tree. Prove by induction that

$$h(n) = \log_2(n+1) - 1$$

Q.3 Can you explain the time complexity for the “parent()” operation when implementing a tree by means of a binary tree?
Q.4 For each of the following traversals, show that you cannot use only its results to reconstruct a unique binary tree. To show this, treat the list of values (a, e, d, b, c, f) as the result of the relevant traversal. In each case, give at least three examples for which the result of traversing the trees is the given list.

(1) inorder traversal
(2) preorder traversal
(3) postorder traversal