Binary Tree Implementation
Lecture 31
Sections 12.2 - 12.3

Robb T. Koether

Hampden-Sydney College

Mon, Apr 6, 2009
Outline

1. The Binary Tree Interface
2. Array Implementation
3. Linked Implementation
4. Assignment
Binary Tree Constructors

- `BinaryTree();`
  Constructs an empty binary tree.

- `BinaryTree(const T& value);`
  Constructs a binary tree with one node with the specified value.

- `BinaryTree(const BinaryTree& lft, const BinaryTree& rgt);`
  Constructs a binary tree with the specified left and right subtrees.
Binary Tree Constructors

- `BinaryTree(const T& value, const BinaryTree& lft, const BinaryTree& rgt);`
  Constructs a binary tree with the specified root value and the specified left and right subtrees.

- `BinaryTree(const BinaryTree& tree);`
  Constructs a copy of an existing binary tree.
Binary Tree Destructor

```cpp
~BinaryTree();
```

Destroys the binary tree.
Binary Tree Inspector's

**int size() const;**
- Returns the number of nodes in the binary tree.

**int height() const;**
- Returns the height of the binary tree.

**bool isEmpty() const;**
- Determines whether the binary tree is empty.

**T& rootValue() const;**
- Returns a reference to the value in the root node.
Binary Tree Inspectors

- `BinaryTree leftSubtree() const;`  
  Returns a copy of the left subtree.

- `BinaryTree rightSubtree() const;`  
  Returns a copy of the right subtree.

- `bool isCountBalanced() const;`  
  Determines whether the binary tree is count balanced.

- `bool isHeightBalanced() const;`  
  Determines whether the binary tree is height balanced.
Binary Tree Mutators

- **void makeEmpty();**
  Removes all the nodes from the binary tree.

- **void setRootValue(const T& value);**
  Sets the root value to the specified value.
Binary Tree Facilitators

- **void input(istream& in);**
  Reads a binary tree from the input stream.
- **void output(ostream& out) const;**
  Writes a binary tree to the output stream.
- **bool isEqual(BinaryTree tree) const;**
  Determines whether two binary trees are equal.
Binary Tree Operators

- `BinaryTree& operator=(const BinaryTree& t);`
  Assigns a binary tree.

- `istream& operator>>(istream& in, BinaryTree& t);`
  Reads a binary tree from the input stream.

- `ostream& operator<<(ostream& out, const BinaryTree& t);`
  Writes a binary tree to the output stream.

- `bool operator==(BinaryTree& t);`
  Determines whether two binary trees are equal.
Binary Tree Traversal Functions

- `void preorderTraversal(void (*visit)(BinaryTreeNode*)) const;`  
  Performs a pre-order traversal of the binary tree.

- `void inorderTraversal(void (*visit)(BinaryTreeNode*)) const;` 
  Performs an in-order traversal of the binary tree.

- `void postorderTraversal(void (*visit)(BinaryTreeNode*)) const;`  
  Performs a post-order traversal of the binary tree.
Binary Tree Traversal Functions

```c
void levelorderTraversal(void (*visit)(BinaryTreeNode*)) const;
```

Performs a level-order traversal of the binary tree.
Other Binary Tree Functions

- **T* search(const T& value) const;**
  Searches the binary tree for a specified value.

- **void draw() const;**
  Draws a representation of the binary tree.
Array Implementation of a Binary Tree

- In an array binary tree, the nodes of the tree are stored in an array.
- Position 0 is left empty.
- The root is stored in position 1.
- For the element in position \( n \),
  - The left child is in position \( 2n \).
  - The right child is in position \( 2n + 1 \).
- For the element in position \( n \), the parent is in position \( n/2 \).
Array Implementation

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
</tr>
</thead>
</table>

10 20 30 40 50 60 70
# Array Implementation

The Binary Tree Interface

## Array Implementation

<table>
<thead>
<tr>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
</tr>
</thead>
</table>

Unused
Array Implementation

Root

10 20 30 40 50 60 70
Array Implementation

Parent

| 10 | 20 | 30 | 40 | 50 | 60 | 70 |
Parents, do you know where your children are?
Array Implementation

- Yes, they are at $2n$ and $2n + 1$. 
Array Implementation

Children, do you know where your parents are?
Yes, Mom and Dad are at $\text{floor}(n/2)$.
Array Implementation

Children

10  20  30  40  50  60  70
Advantages of the Array Implementation

- This representation is very efficient when
  - The tree is complete, and
  - The structure of the tree will not be modified.
As we have seen, the linked implementation uses `BinaryTreeNode`s.

Each `BinaryTreeNode` has two node pointers, one to the left subtree and one to the right subtree.

The `BinaryTree` itself consists of a single node pointer to the root node.
Linked Binary Tree Implementation

**Constructor**

- `BinaryTree(const T& value, const BinaryTree& lft, const BinaryTree& rgt);

- Implement the above constructor.
Linked Binary Tree Implementation

The Destructor and `makeEmpty()`

- `~BinaryTree();`
- `void makeEmpty();`

Implement the destructor along with the recursive and non-recursive `makeEmpty()` functions.
makeCopy()

- **void** makeCopy(BinaryTreeNode* & new_node,
  const BinaryTreeNode* old_node);

- Implement the private, recursive function makeCopy().
Assignment

Homework

- Read Section 12.2, pages 651 - 658.
- Read Section 12.3, pages 660 - 664.