Binary Tree Implementation

Lecture 31
Sections 12.2 - 12.3

Robb T. Koether
Hampden-Sydney College

Mon, Apr 5, 2010
Outline

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

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

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BinaryTree()</td>
<td>Constructs an empty binary tree.</td>
</tr>
<tr>
<td>BinaryTree(const T&amp; value)</td>
<td>Constructs a binary tree with one node with the specified value.</td>
</tr>
<tr>
<td>BinaryTree(const BinaryTree&amp; lft, const BinaryTree&amp; rgt)</td>
<td>Constructs a binary tree with the specified left and right subtrees.</td>
</tr>
</tbody>
</table>

- **BinaryTree()** – Constructs an empty binary tree.
- **BinaryTree(T)** – Constructs a binary tree with one node with the specified value.
- **BinaryTree(BinaryTree, BinaryTree)** – Constructs a binary tree with the specified left and right subtrees.
Binary Tree Constructors

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

- `BinaryTree(T, BinaryTree, BinaryTree)` – Constructs a binary tree with the specified root value and the specified left and right subtrees.
- `BinaryTree(BinaryTree)` – Constructs a copy of an existing binary tree.
Binary Tree Destructor

~BinaryTree();

~BinaryTree() – Destroys the binary tree.
Binary Tree Inspectors

```cpp
int size() const;
int height() const;
bool isEmpty() const;
T rootValue() const;
T& rootValue();
```

- `size()` – Returns the number of nodes in the binary tree.
- `height()` – Returns the height of the binary tree.
- `isEmpty()` – Determines whether the binary tree is empty.
- `rootValue()` `const` – Returns a copy the value in the root node.
- `rootValue()` – Returns a reference to the value in the root node.
## Binary Tree Inspectors

- `leftSubtree()` – Returns a copy of the left subtree.
- `rightSubtree()` – Returns a copy of the right subtree.
- `isCountBalanced()` – Determines whether the binary tree is count balanced.
- `isHeightBalanced()` – Determines whether the binary tree is height balanced.
Binary Tree Mutators

```c
void makeEmpty();
```

- `makeEmpty()` – Removes all the nodes from the binary tree.
Binary Tree Facilitators

void input(istream& in);
void output(ostream& out) const;
bool isEqual(BinaryTree tree) const;

- **input()** — Reads a binary tree from the input stream.
- **output()** — Writes a binary tree to the output stream.
- **isEqual()** — Determines whether two binary trees are equal.
Binary Tree Operators

- **operator=()** – Assigns a binary tree.
- **operator>>( )** – Reads a binary tree from the input stream.
- **operator<<( )** – Writes a binary tree to the output stream.
- **operator==()** – Determines whether two binary trees are equal.
- **operator!=( )** – Determines whether two binary trees are not equal.
Binary Tree Traversal Functions

- **preorderTraversal()** – Performs a pre-order traversal of the binary tree.
- **inorderTraversal()** – Performs an in-order traversal of the binary tree.
- **postorderTraversal()** – Performs a post-order traversal of the binary tree.
- **levelorderTraversal()** – Performs a level-order traversal of the binary tree.
Other Binary Tree Functions

- `search(const T& value) const`;
- `void draw() const`;

- `search()` — Searches the binary tree for a specified value.
- `draw()` — Draws a representation of the binary tree.
Outline

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

| 10 | 20 | 30 | 40 | 50 | 60 | 70 |
Array Implementation

Unused

10  20  30  40  50  60  70
Array Implementation

Root

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

Parent

| 10 | 20 | 30 | 40 | 50 | 60 | 70 |

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

Children, do you know where your parents are?
Array Implementation

Yes, Mom and Dad are at \( \text{floor}(n/2) \).
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.
Outline

1. The Binary Tree Interface
2. Array Implementation
3. Linked Implementation
4. Assignment
As we have seen, the linked implementation uses \texttt{BinaryTreeNode}s.

Each \texttt{BinaryTreeNode} has two node pointers, one the the left subtree and one to the right subtree.

The \texttt{BinaryTree} itself consists of a single node pointer to the root node.
**Constructor**

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

- Implement the above constructor.
The Destructor and makeEmpty()

void makeEmpty();

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

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

- Implement the private, recursive function `makeCopy()`.
Homework

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