

Binary Tree Implementation

Lecture 31
Section 19.1

Robb T. Koether

Hampden-Sydney College

Mon, Apr 10, 2017

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

Binary Tree Constructors

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

- `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

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

Binary Tree Destructor

```
~BinaryTree();
```

- `~BinaryTree()` – Destroys the binary tree.

Binary Tree Inspectors

Binary Tree Inspectors

```
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

Binary Tree Inspectors

```
BinaryTree leftSubtree() const;  
BinaryTree rightSubtree() const;  
bool isCountBalanced() const;  
bool isHeightBalanced() const;
```

- `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

Binary Tree Mutators

```
void rootValue(const T& value);  
    void makeEmpty();
```

- `rootValue()` – Assigns the value to the root node.
- `makeEmpty()` – Removes all the nodes from the binary tree.

Binary Tree Facilitators

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

Binary Tree Operators

```
BinaryTree& operator=(const BinaryTree& t);  
istream& operator>>(istream& in, BinaryTree& t);  
ostream& operator<<(ostream& out, const BinaryTree& t);  
bool operator==(const BinaryTree& t1, const BinaryTree& t2);  
bool operator!=(const BinaryTree& t1, const BinaryTree& t2);
```

- **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

Binary Tree Traversal Functions

```
void preorderTraversal(void (*visit)(BinaryTreeNode*)) const;  
void inorderTraversal(void (*visit)(BinaryTreeNode*)) const;  
void postorderTraversal(void (*visit)(BinaryTreeNode*)) const;  
void levelorderTraversal(void (*visit)(BinaryTreeNode*)) const;
```

- `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

Other Binary Tree Functions

```
T* 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

Array Implementation of a Binary Tree

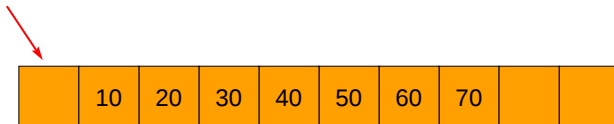
- In an array binary tree, the nodes of the tree are stored in an array.
- Position 0 is left empty.
- The root is stored in position 1.
- For the element in position n ,
 - The left child is in position $2n$.
 - The right child is in position $2n + 1$.
- For the element in position n , the parent is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

Array Implementation

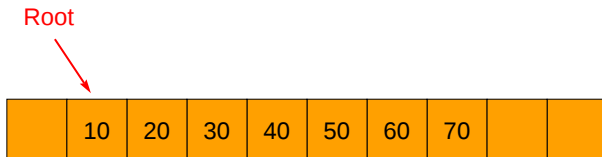
	10	20	30	40	50	60	70		
--	----	----	----	----	----	----	----	--	--

Array Implementation

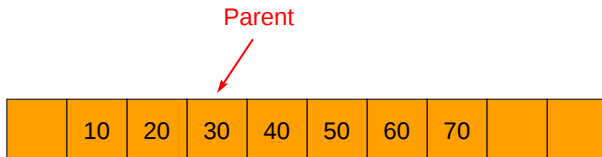
Unused



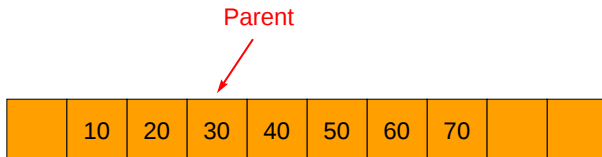
Array Implementation



Array Implementation

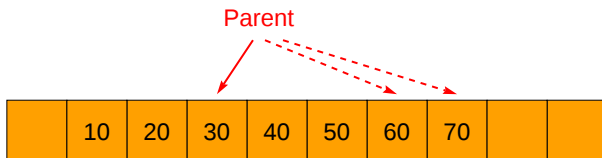


Array Implementation



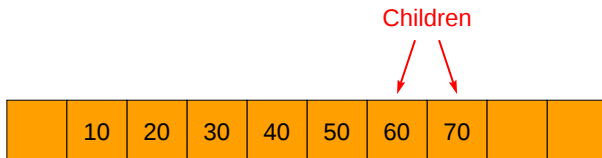
Parents, do you know where your children are?

Array Implementation



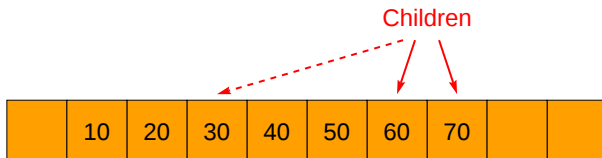
Yes, they are at $2n$ and $2n + 1$.

Array Implementation



Children, do you know where your parents are?

Array Implementation



Yes, Mom and Dad are at `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.
- Otherwise, it is better to use a dynamic (linked) structure.

Outline

1 The Binary Tree Interface

2 Array Implementation

3 Linked Implementation

4 Assignment

Linked Binary Tree Implementation

- As we have seen, the linked implementation uses `BinaryTreeNodeS`.
- 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.

Linked Binary Tree Implementation

makeCopy()

```
void makeCopy(const BinaryTree<T>& tree);  
BinaryTreeNode<T>* makeTree(  
    const BinaryTreeNode<T>* node) const;
```

- `makeTree()` – Makes a copy of the tree whose root node is the specified node and returns a pointer to the root node of the copy.
- Implement the nonrecursive function `makeCopy()` and the recursive function `makeTree()`.

Outline

1 The Binary Tree Interface

2 Array Implementation

3 Linked Implementation

4 Assignment

Assignment

Assignment

- Read Section 19.1.