CS 535 Object-Oriented Programming & Design
Fall Semester, 2000
Doc 7 Distributing System Intelligence
Contents

Distributing System Intelligence.....2
Solution 0 SmartTree, StructNode 3
Solution 1 SmartTree, DumbNode 4
Solution 2 DumbTree, BSTNode.....9
DumbTree.....11
TreeNode.....12
BSTNode 13
NilLeaf.....15
How Does this Work? 16
Comparison.....20
Avoid Case (and if) Statements 21
Issues: Performance.....22

Reference:

The Art of Computer Programming, Vol 3 Sorting and Searching, Knuth

Distributing System Intelligence
A Tree Example

Problem:

Implement a binary search tree with operations:

put(int key, Object value)

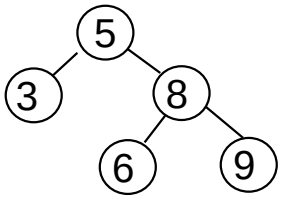
Puts the specified value into the tree, using the specified key.

get(int key)

Gets the object associated with the specified key in the tree.

toString()

Returns a string representation of the tree
(leftSubtree root rightSubtree)



((3)5((6)8(9)))

Solution 0 SmartTree, StructNode

StructNode

```
class StructNode
   {
   public StructNode left = null;
   public StructNode right = null;

   public int key;
   public Object value;
   }
```

Solution 1 SmartTree, DumbNode

DumbNode

```
class DumbNode
   {
   protected DumbNode left = null;
   protected DumbNode right = null;

   protected int key;
   protected Object value;

   public DumbNode( int key, Object value )
      {
      this.key = key;
      this.value = value;
      }

   }
```

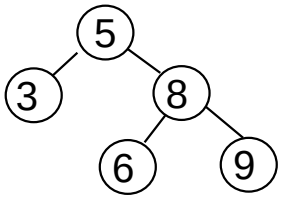
SmartTree Fields

```
package sdsu.trees;

import java.util.*;

public class SmartTree
   {
   protected DumbNode root = null;

   // Methods shown later
   }
```

Since everyone has implemented a tree search I do not show put and get methods

SmartTree Methods
Preorder Traversal

1) Print "(" then Visit left subtree
2) Print node
3) Visit right subtree, then print ")"



Applying rule we get:

(left subtree 5 right subtree)
((3) 5 (left subtree 8 right subtree))
((3) 5 ((6) 8 (9)))

SmartTree Methods
toString() Helper Class

The following three pages implement a standard algorithm to perform a preorder traversal of a binary tree. Check any data structures text. You will find this implementation in the book. The point is to show how long and complex this is. You do not have to understand it.

Need to store path of nodes visited on a stack with which visit we are on: first, second or third

```
public class TraversalInfo
   {
   public DumbNode node;
   public int visitNumber;

   public TraversalInfo( DumbNode nodeTraversed, int visitNumber )
      {
      node = nodeTraversed;
      this.visitNumber = visitNumber;
      }

   }
```

Constants for toString()

```
private final static int FIRST = 1;
private final static int SECOND = 2;
private final static int THIRD = 3;
```

SmartTree Methods
toString(): Simple Algorithm

```
public String toString()
  {
  StringBuffer treeString = new StringBuffer();
  TraversalInfo currentLocation;

  Stack visited = new Stack();
  visited.push( new TraversalInfo( root, FIRST ) );

  while ( visited.empty() != true )
    {
    currentLocation = (TraversalInfo) visited.pop();

    switch ( currentLocation.visitNumber )
      {
      case FIRST:
        treeString.append( "(" );
        firstVisit( visited, currentLocation );
        break;

      case SECOND:
        treeString.append( currentLocation.node.key );
        secondVisit( visited, currentLocation );
        break;

      case THIRD:
        treeString.append( ")" );
        break;
      }
    }
  return treeString.toString();
  }
```

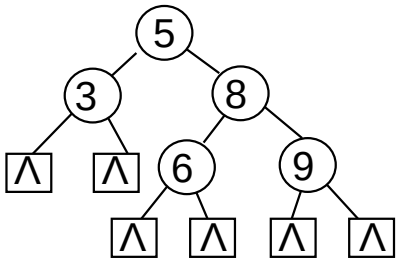
SmartTree Methods
toString(): continued

```
protected void firstVisit( Stack visited, TraversalInfo currentLocation
)
   {
   DumbNode nextnode;
   currentLocation.visitNumber = SECOND;
   visited.push( currentLocation );

   if ( currentLocation.node.left != null )
      {
      nextnode = currentLocation.node.left;
      visited.push( new TraversalInfo( nextnode, FIRST )  );
      }
   }

protected void secondVisit( Stack visited,
                        TraversalInfo currentLocation )
   {
   DumbNode nextnode;
   currentLocation.visitNumber = THIRD;
   visited.push( currentLocation );

   if ( currentLocation.node.right != null )
      {
      nextnode = currentLocation.node.right;
      visited.push( new TraversalInfo( nextnode, FIRST )  );
      }
   }
```

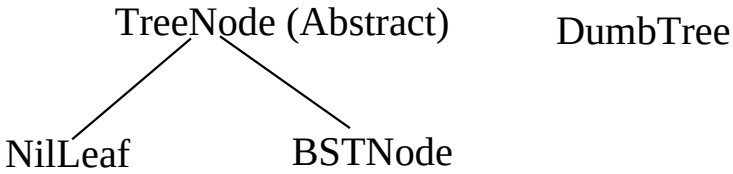
Solution 2 DumbTree, BSTNode

1) Let the nodes do some work

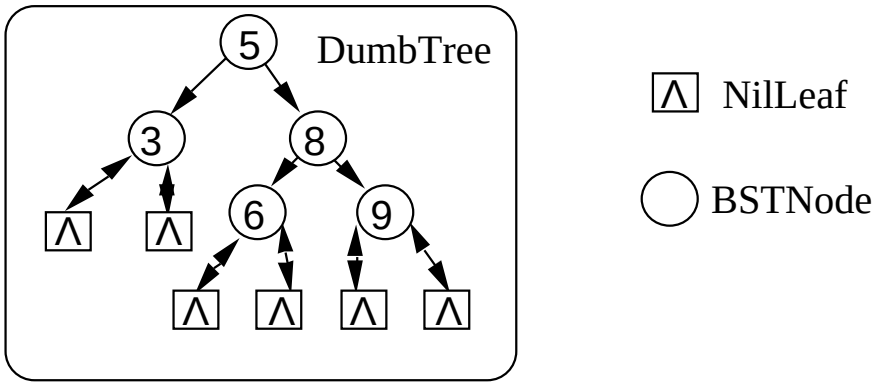
2) Add some nil leaves to eliminate some cases



Class Structure
Inheritance



Runtime Structure



Preorder Traversal - Basic Idea

BSTNode

Get the preorder of left subtree, right subtree,
Put key in middle

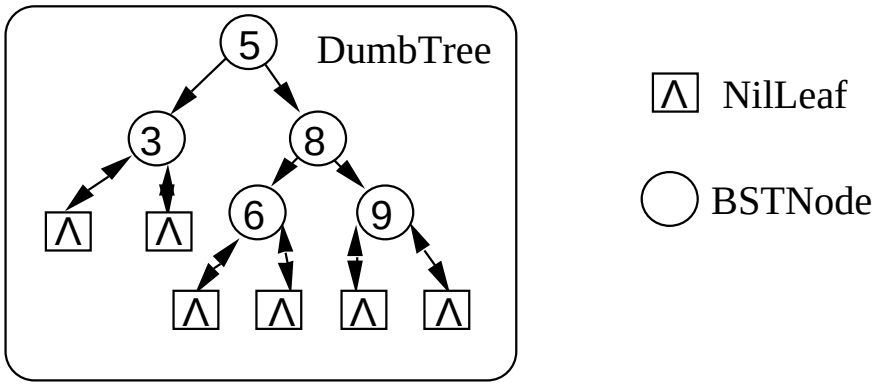
```
public String toString()
   {
   return "(" + left.toString()  + key +  right.toString() + ")";
   }
```

NilLeaf

Represents an empty node

Action is to do nothing

```
public String toString()
   {
   return "";
   }
```



DumbTree

```
package sdsu.trees;

public class DumbTree
  {
  protected TreeNode root = null;

  public Object get( int key )
    {
    if ( root == null )
      return null;

    return root.getNode( key ).value();
    }

  public Object put( int key, Object value )
    {
    if ( root == null )
      {
      root = new BSTNode( key, value );
      return null;
      }

    return root.getNode( key ).put( key, value );
    }

  public String toString()
    {
    return root.toString();
    }
  }
```

TreeNode

```
abstract class TreeNode
  {

  /**
   * Puts the specified key & value in this node
   */
  abstract public Object put( int key, Object value );

  /**
   * Return the value of the TreeNode
   */
  abstract public Object value();


  /**
   * If keyToFind is in the subtree rooted at this node, then return
   * node containing keyToFind.
   * Otherwise return the NilLeaf that should contain keyToFind
   */
  abstract public TreeNode getNode( int key );


  /**
   * Return an ascii representation of tree rooted at this node
   */
  abstract public String toString();
  }
```

Comments

There is no common code or methods between BSTNode and NilLeaf

TreeNode could be either an interface or an abstract class

BSTNode

```
class BSTNode extends TreeNode
   {
   protected TreeNode left;
   protected TreeNode right;

   protected int key;
   protected Object value;

   public BSTNode( int key, Object value )
      {
      this.key = key;
      this.value = value;
      left = new NilLeaf( this );
      right = new NilLeaf( this );
      }

   /**
    * Return the value of the TreeNode
    */
   public Object value()
      {
      return value;
      }

   /**
    * Return an ascii representation of tree rooted at this node
    */
   public String toString()
      {
      return "(" + left.toString()  + key +  right.toString() + ")";
      }
```

Note how simple the toString() method is here!

BSTNode Continued

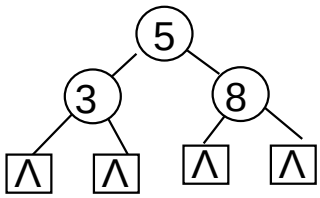
```
public Object put( int keyToAdd, Object valueToAdd )
   {
   Object oldValue = value;
   value = valueToAdd;
   return oldValue;
   }

/**
 * If keyToFind is in the subtree rooted at this node, then return
 * node containing keyToFind.
 * Otherwise return the NilLeaf that should contain keyToFind
 */
public TreeNode getNode( int keyToFind )
   {
   if ( keyToFind < key )
      return left.getNode( keyToFind );
   else if ( keyToFind > key )
      return right.getNode( keyToFind );
   else
      return this;
   }

/**
 * Puts indicated key and value in proper child of this node
 */
protected void putAsChild( int keyToAdd, Object valueToAdd )
   {
   if ( keyToAdd < key )
      left = new BSTNode( keyToAdd, valueToAdd );
   else if ( keyToAdd > key )
      right = new BSTNode( keyToAdd, valueToAdd );
   }
}
```

NilLeaf

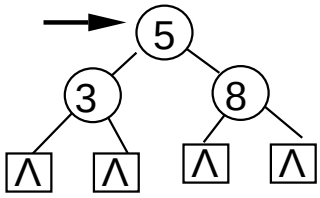
```
class NilLeaf extends TreeNode
  {
  protected BSTNode parent;

  public NilLeaf( BSTNode parent )
    {
    this.parent = parent;
    }

  public Object put( int key, Object value )
    {
    parent.putAsChild( key, value );
    return null;
    }

  public Object value()
    {
    return null;
    }

  public TreeNode getNode( int key )
    {
    return this;
    }

  public String toString()
    {
    return "";
    }
  }
```

How Does this Work?

```
DumbTree example = new DumbTree();
example.put( 5, null );
example.put( 3, null );
example.put( 8, null );
```

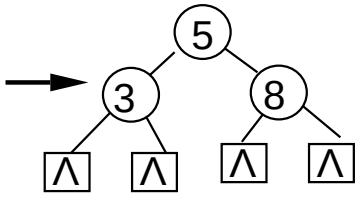



```
// Now add a 1

example.put( 1, null );

// In DumbTree's put( 1, null ) method does:

   return root.getNode( 1 ).put( 1, null );
```



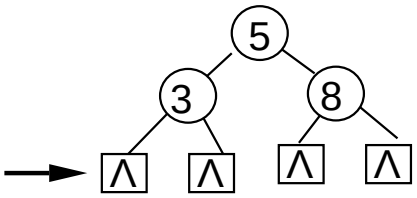

```
// in BSTNode with key 5 method getNode( 1 ) does:

      if ( 1 < 5 )
         return left.getNode( 1 );
      else if ( 1 > 5 )
         return right.getNode( 1 );
      else
         return this;
```

Example Continued



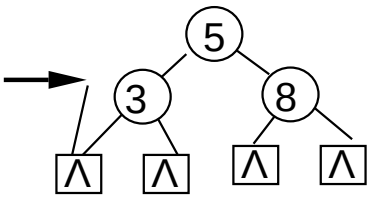
// in BSTNode with key 3 method getNode(1) does:

```
if ( 1 < 3 )
   return left.getNode( 1 );
else if ( 1 > 3 )
```



// in NilNode method getNode(1) does:

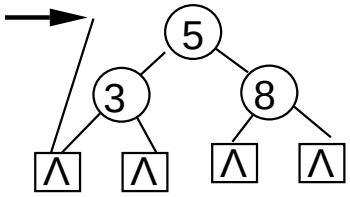
```
return this;
```



// in BSTNode with key 3 method getNode(1) does:

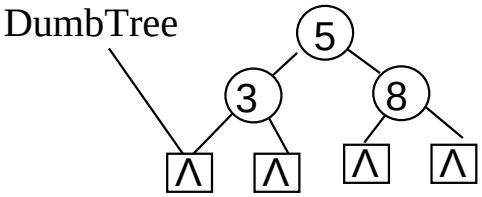
```
if ( 1 < 3 )
   return left.getNode( 1 );
```

Example Continued



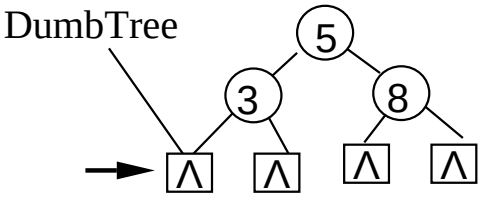
// in BSTNode with key 5 method getNode(1) does:

```
if ( 1 < 5 )
   return left.getNode( 1 );
```



// In DumbTree's put(1, null) method does:

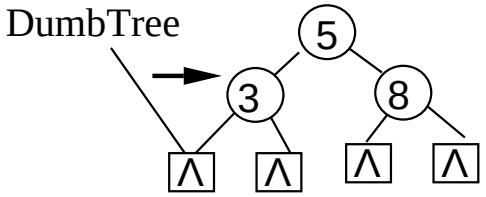
```
return root.getNode( 1 ).put( 1, null );
```



// in NilNode method put(1, null) does:

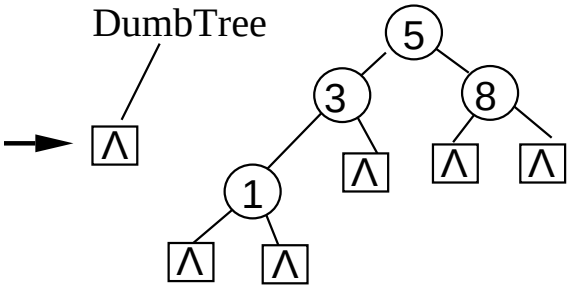
```
parent.putAsChild( 1, null );
return null;
```

Example Continued



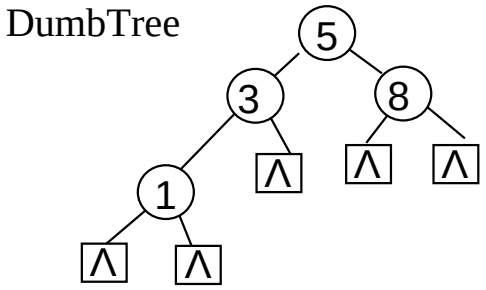
// in BSTNode with key 3 method putAsChild(1, null) does:

```
if ( 1 < 3 )
  left = new BSTNode( 1, null );
else if ( 1 > 3 )
```



// in NilNode method put(1, null) does:

```
parent.putAsChild( 1, null );
return null;
```



// In DumbTree's put(1, null) method does:

```
return root.getNode( 1 ).put( 1, null );
```

Comparison

Metric	SmartTree	DumbTree
LOC[1]	60	36
Number of classes	3	4
Number of methods	6[2]	18[3]
LOC/method	10	2

SmartTree is centralized

DumbTree distributes the logic in the tree structure

[1]What is a line of code (LOC)? Do you count a "{" as a line of code? For this comparison I selected a method of counting made easy by my word processor: I counted the number of ";" in the code.
[2]Originally firstVisit and second visit methods did not exist. They were added just to make each method fit on one slide. Hence I did not count them as methods for this comparison.
[3]This includes the abstract class

Issue
Avoid Case (and if) Statements

Implementation that avoids if statements by sending a message to an object

NilLeaf returns a null string;

```
public String toString()
   {
   return "(" + left.toString()  + key +  right.toString() + ")";
   }
```

Implementation that uses if statements

```
public String toString()
   {
   String treeRepresentation;

   treeRepresentation = "(";

   if ( left != null )
      treeRepresentation = treeRepresentation + left.toString();

   treeRepresentation = treeRepresentation + left.toString();

   if ( right != null )
      treeRepresentation = treeRepresentation + right.toString();

   treeRepresentation = treeRepresentation + ")";

   return treeRepresentation;
   }
```

Issues: Performance

Have you lost your mind?

- NilLeaf doubles the space requirement
- DumbTree's recursive like search requires considerable stack space
- DumbTree's recursive like search is slower than SmartTree's iterative search

NilLeaf doubles the space requirement

True, but one only needs NilLeaf per tree

Recursion requires considerable stack space

This is true of any recursive solution

Simulations indicate Metroworks Java implementation runs out of stack space after about 8,200 recursive calls to the same method

See AVLTree

DumbTree is slower than SmartTree
Performance Test

Insert ints from 1 to N into each tree

Look up each int once

Times are measured on a PowerMac 7100/80

Times are in milliseconds

Timing Results

N ->	400	800	1600
SmartTree create	264	1064	4342
DumbTree create	314	1290	5681
SmartTree find all	261	1049	4197
DumbTree find all	272	1193	5370

More Timing Results

N ->	400	800	1600
SmartTree create	264	1064	4342
DumbTree create	314	1290	5681
DumbAVLTree create	46	94	196
Hashtable create	42	82	165
Opt. Hashtable create	25	57	102
SmartTree find all	261	1049	4197
DumbTree find all	272	1193	5370
DumbAVLTree find all	11	23	50
Hashtable find all	30	61	127
Opt. Hashtable find all	14	27	54

SmartTree is a binary search tree using iterative search

DumbTree is a binary search tree using recursive search

DumbAVLTree is an AVL tree using recursive search

Hashtable is Java's standard Hashtable with methods synchronized

Opt. Hashtable is Java's standard Hashtable with synchronization removed