Recursive Tree Traversals

- In-order

```
private void printInOrder(TreeNode node) {
   if (node != null) {
      printInOrder(node.getLeft());
      System.out.print(node.getItem() + " ");
      printInOrder(node.getRight());
   }
}
```

Write the pre-order and post-order versions.

© 2006 Pearson Addison-Wesley. All rights reserved

The ADT Binary Search Tree

- A deficiency of the ADT binary tree which is corrected by the ADT binary search tree
  - Searching for a particular item
- Each node n in a binary search tree satisfies the following properties
  - n's value is greater than all values in its left subtree T_L
  - n's value is less than all values in its right subtree T_R
  - Both T_L and T_R are binary search trees

© 2006 Pearson Addison-Wesley. All rights reserved

The ADT Binary Search Tree

- Record
  - A group of related items, called fields, that are not necessarily of the same data type
- Field
  - A data element within a record
- A data item in a binary search tree has a specially designated search key
  - A search key is the part of a record that identifies it within a collection of records
- `KeyedItem` class
  - Contains the search key as a data field and a method for accessing the search key – `getKey()`
  - Must be extended by classes for items that are in a binary search tree

© 2006 Pearson Addison-Wesley. All rights reserved

The ADT Binary Search Tree

- Operations of the ADT binary search tree
  - Insert a new item into a binary search tree
  - Delete the item with a given search key from a binary search tree
  - Retrieve the item with a given search key from a binary search tree
  - Traverse the items in a binary search tree in preorder, inorder, or postorder

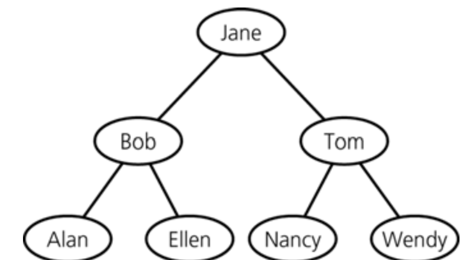


Figure 11-19

A binary search tree

© 2006 Pearson Addison-Wesley. All rights reserved

Simplified Binary Search Tree TreeNode class

```
public class TreeNode {

    private KeyedItem item; // has a getKey method
    private TreeNode left;
    private TreeNode right;

    public Comparable getKey() {
        return item.getKey();
    }
}
```

Plus all other accessor methods and constructors.

© 2006 Pearson Addison-Wesley. All rights reserved

Algorithms for the Operations of the ADT Binary Search Tree

- Searching and retrieving from a BST

```
private Item search(TreeNode node, Comparable key) {
    if (node == null)
      return null;
    else {
        int comparison = key.compareTo(node.getKey());
        if (comparison < 0)
          return search(node.getLeft(), key);
        else if (comparison > 0)
          return search(node.getRight(), key);
        else
          return node.getItem();
    }
}
```

© 2006 Pearson Addison-Wesley. All rights reserved

Algorithms for the Operations of the ADT Binary Search Tree: Insertion

- `insert(treeNode, newItem)`
  - Inserts `newItem` into the binary search tree of which `treeNode` is the root

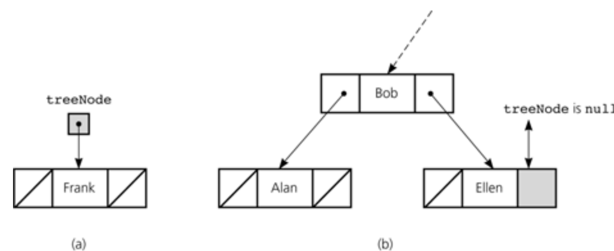


Figure 11-23a and 11-23b

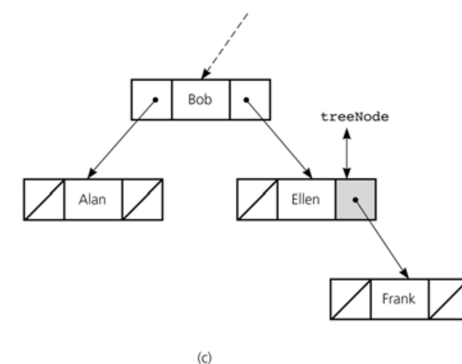
a) Insertion into an empty tree; b) search terminates at a leaf

© 2006 Pearson Addison-Wesley. All rights reserved

Algorithms for the Operations of the ADT Binary Search Tree: Insertion

Figure 11-23c

c) insertion at a leaf



© 2006 Pearson Addison-Wesley. All rights reserved

Simplified insertion code

```
private TreeNode insert(TreeNode node, Item newItem) {

    if (node == null)
      node = new TreeNode(newItem);
    else if (newItem.getKey().compareTo(node.getKey()) < 0)
      node.setLeft(insert(node.getLeft(), newItem));
    else
      node.setRight(insert(node.getRight(), newItem));
    return node;

}
```

© 2006 Pearson Addison-Wesley. All rights reserved

Algorithms for the Operations of the ADT Binary Search Tree: Deletion

- Steps for deletion
  - search for the item with the specified key
  - If the item is found, remove the item from the tree
- Three possible cases for node N containing the item to be deleted
  - N is a leaf
  - N has only one child
  - N has two children

© 2006 Pearson Addison-Wesley. All rights reserved

Algorithms for the Operations of the ADT Binary Search Tree: Deletion

- Strategies for deleting node N
  - If N is a leaf
    - Set the reference in N's parent to `null`
  - If N has only one child
    - Let N's parent adopt N's child
  - If N has two children
    - Locate another node M that is easier to remove from the tree than the node N, the inorder successor (see page 565)
    - Copy the item that is in M to N
    - Remove the node M from the tree

© 2006 Pearson Addison-Wesley. All rights reserved

Algorithms for the Operations of the ADT Binary Search Tree: Traversal

- Traversals for a binary search tree are the same as the traversals for a binary tree
- Theorem 11-1

  The inorder traversal of a binary search tree T will visit its nodes in sorted search-key order

© 2006 Pearson Addison-Wesley. All rights reserved

The Efficiency of Binary Search Tree Operations

- The maximum number of comparisons for a retrieval, insertion, or deletion is the height of the tree
- The maximum and minimum heights of a binary search tree
  - n is the maximum height of a binary tree with n nodes

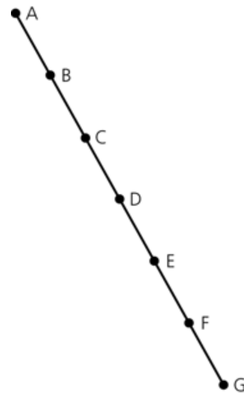


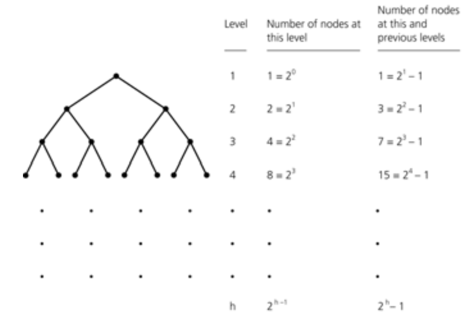
Figure 11-30
A maximum-height binary tree with seven nodes

© 2006 Pearson Addison-Wesley. All rights reserved

The Efficiency of Binary Search Tree Operations

- Theorem 11-2
  A full binary tree of height h has $2^h - 1$ nodes
- Theorem 11-3
  The maximum number of nodes that a binary tree of height h can have is $2^h - 1$

Figure 11-32
Counting the nodes in a full binary tree of height *h*



© 2006 Pearson Addison-Wesley. All rights reserved

The Efficiency of Binary Search Tree Operations

- Theorem 11-4
  The minimum height of a binary tree with n nodes is $\lceil \log_2(n+1) \rceil$
- The height of a particular binary search tree depends on the order in which insertion and deletion operations are performed

Operation	Average case	Worst case
Retrieval	O(log n)	O(n)
Insertion	O(log n)	O(n)
Deletion	O(log n)	O(n)
Traversal	O(n)	O(n)

Figure 11-34
The order of the retrieval, insertion, deletion, and traversal operations for the reference-based implementation of the ADT binary search tree

© 2006 Pearson Addison-Wesley. All rights reserved

Treesort

- Treesort
  - Uses the ADT binary search tree to sort an array of records into search-key order
  - Efficiency
    - Average case: O(n * log n)
    - Worst case: $O(n^2)$

© 2006 Pearson Addison-Wesley. All rights reserved

Saving a Binary Search Tree in a File

- Two algorithms for saving and restoring a binary search tree
  - Saving a binary search tree and then restoring it to its original shape
    - Uses preorder traversal to save the tree to a file
  - Saving a binary tree and then restoring it to a balanced shape
    - Uses inorder traversal to save the tree to a file
    - Can be accomplished if
      - The data is sorted
      - The number of nodes in the tree is known

© 2006 Pearson Addison-Wesley. All rights reserved

General Trees

- An n-ary tree
  - A generalization of a binary tree whose nodes each can have no more than n children

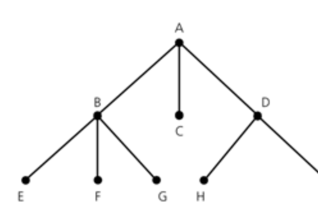


Figure 11-38

A general tree

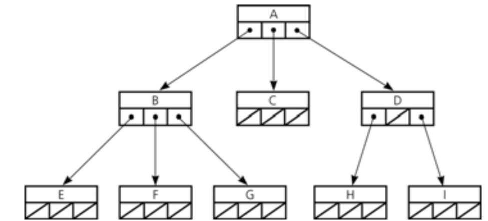


Figure 11-41

An implementation of the *n*-ary tree in Figure 11-38

© 2006 Pearson Addison-Wesley. All rights reserved

Summary

- Binary trees provide a hierarchical organization of data
- Implementation of binary trees
  - The implementation of a binary tree is usually referenced-based
  - If the binary tree is complete, an efficient array-based implementation is possible
- Traversing a tree is a useful operation
- The binary search tree allows you to use a binary search-like algorithm to search for an item with a specified value

© 2006 Pearson Addison-Wesley. All rights reserved

Summary

- Binary search trees come in many shapes
  - The height of a binary search tree with n nodes can range from a minimum of $\lceil \log_2(n + 1) \rceil$ to a maximum of n
  - The shape of a binary search tree determines the efficiency of its operations
- An inorder traversal of a binary search tree visits the tree's nodes in sorted search-key order
- The treesort algorithm efficiently sorts an array by using the binary search tree's insertion and traversal operations

© 2006 Pearson Addison-Wesley. All rights reserved

Summary

- Saving a binary search tree to a file
  - To restore the tree as a binary search tree of minimum height
    - Perform inorder traversal while saving the tree to a file
  - To restore the tree to its original form
    - Perform preorder traversal while saving the tree to a file

© 2006 Pearson Addison-Wesley. All rights reserved