Trees

Chapter 15

Content

- Terminology
- The ADT Binary Tree
- The ADT Binary Search Tree

Terminology

- Use trees to represent relationships
- Trees are hierarchical in nature
  - “Parent-child” relationship exists between nodes in tree.
  - Generalized to ancestor and descendant
  - Lines between the nodes are called edges
- A subtree in a tree is any node in the tree together with all of its descendants

Terminology

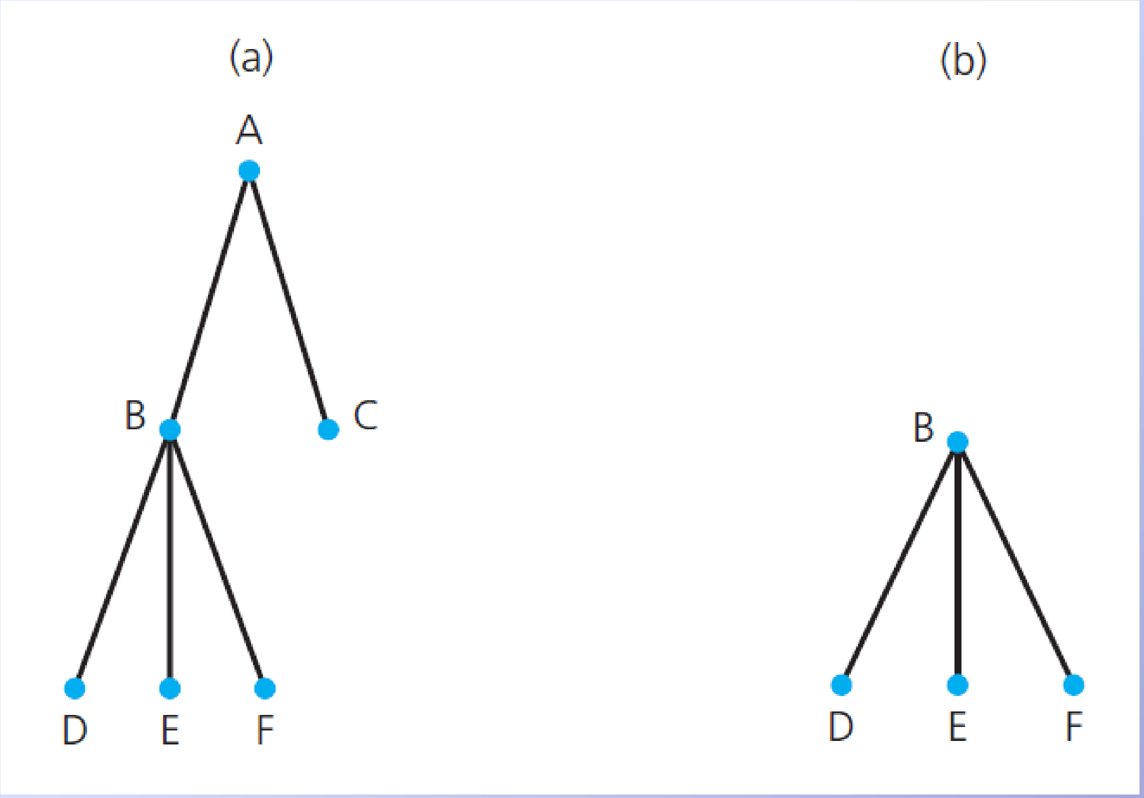


FIGURE 15-1 (a) A tree;
(b) a subtree of the tree in part a

Terminology

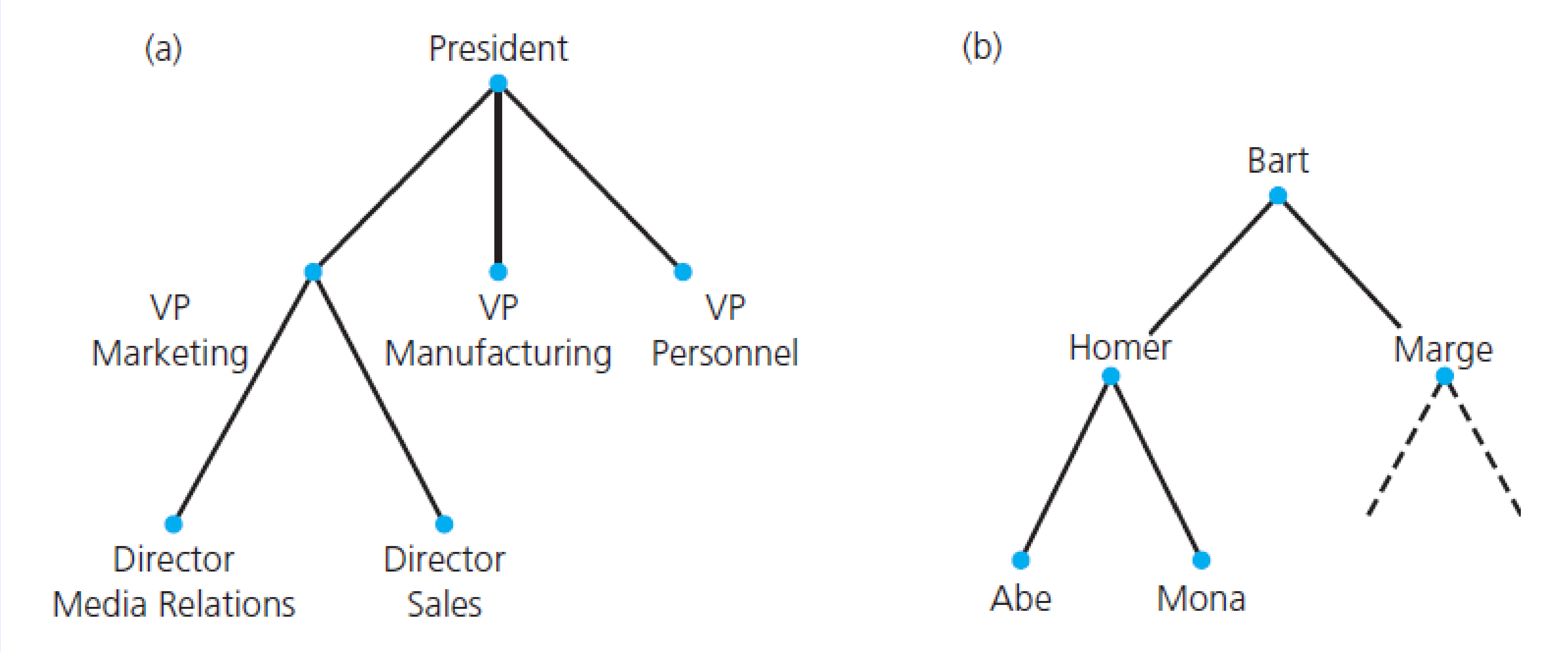


FIGURE 15-2 (a) An organization chart; (b) a family tree

Kinds of Trees

- General Tree
  - Set T of one or more nodes such that T is partitioned into disjoint subsets
  - A single node r , the root
  - Sets that are general trees, called subtrees of r

Kinds of Trees

- n -ary tree
  - set T of nodes that is either empty or partitioned into disjoint subsets:
  - A single node r , the root
  - n possibly empty sets that are n -ary subtrees of r

Kinds of Trees

- Binary tree
  - Set T of nodes that is either empty or partitioned into disjoint subsets
  - Single node r , the root
  - Two possibly empty sets that are binary trees, called left and right subtrees of r

Example: Algebraic Expressions.

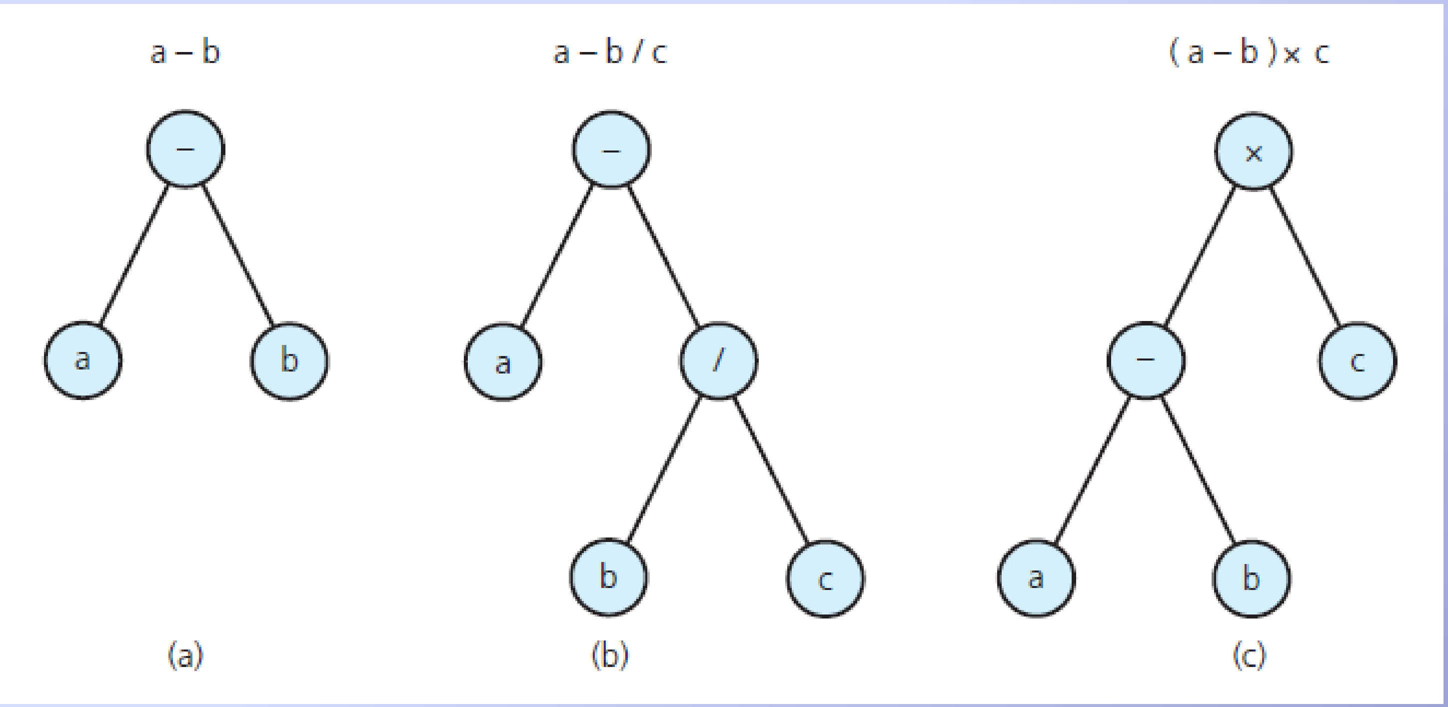


FIGURE 15-3 Binary trees that represent algebraic expressions

Binary Search Tree

- For each node n, a binary search tree satisfies the following three 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.

Binary Search Tree

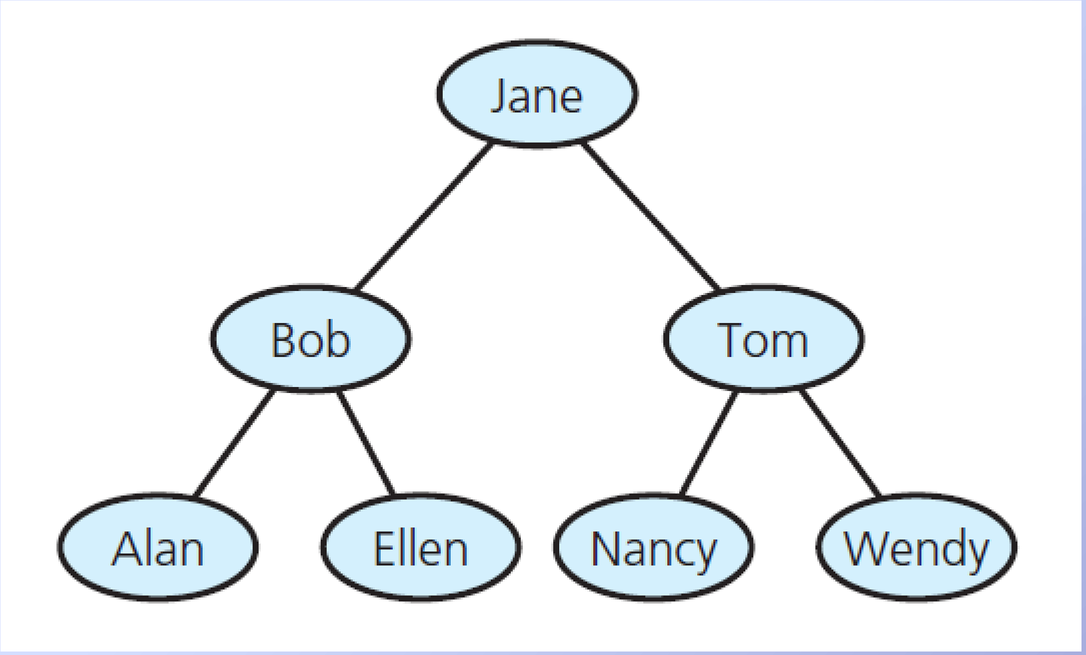


FIGURE 15-4 A binary search tree of names

The Height of Trees

- Definition of the level of a node n :
  - If n is the root of T , it is at level 1.
  - If n is not the root of T , its level is 1 greater than the level of its parent.
- Height of a tree T in terms of the levels of its nodes
  - If T is empty, its height is 0.
  - If T is not empty, its height is equal to the maximum level of its nodes.

The Height of Trees

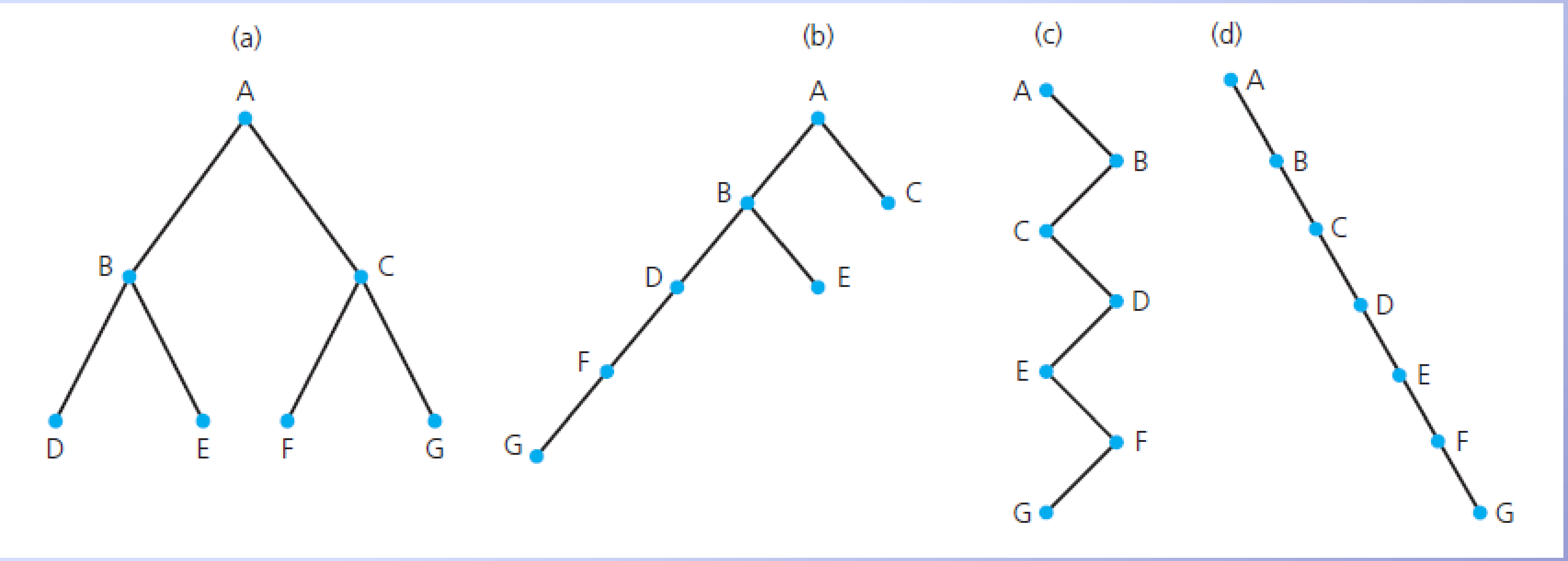


FIGURE 15-5 Binary trees with the same nodes but different heights

Full, Complete, and Balanced Binary Trees

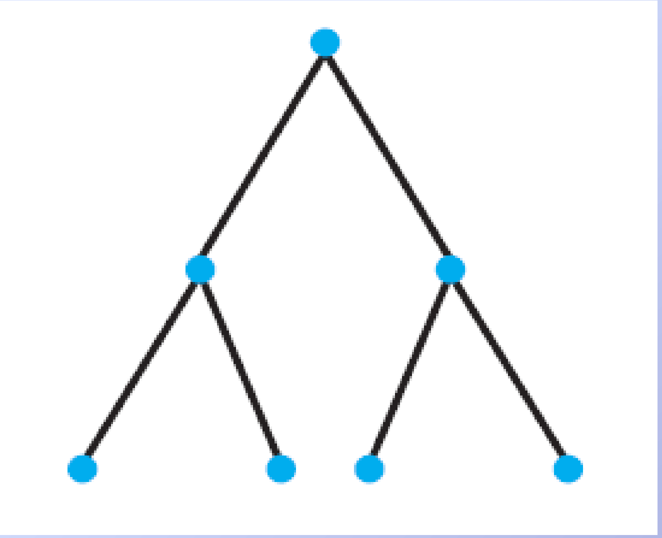


FIGURE 15-6 A full binary tree of height 3

Full, Complete, and Balanced Binary Trees

- Definition of a full binary tree
  - If T is empty, T is a full binary tree of height 0.
  - If T is not empty and has height $h > 0$, T is a full binary tree if its root's subtrees are both full binary trees of height $h - 1$.

Full, Complete, and Balanced Binary Trees

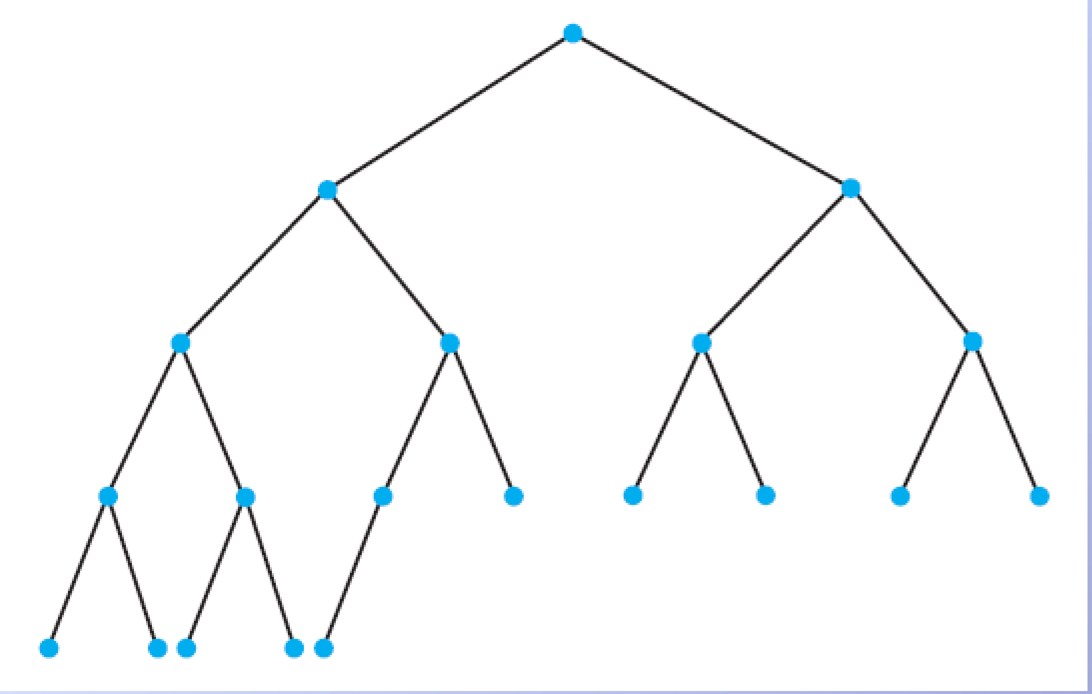


FIGURE 15-7 A complete binary tree

The Maximum and Minimum Heights of a Binary Tree

- The maximum height of an n -node binary tree is n .

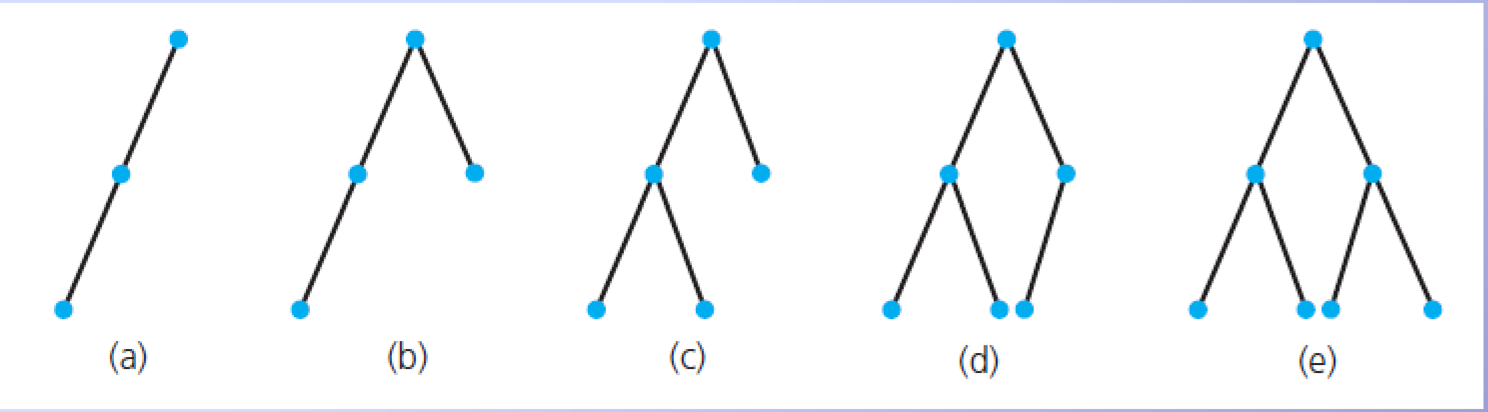
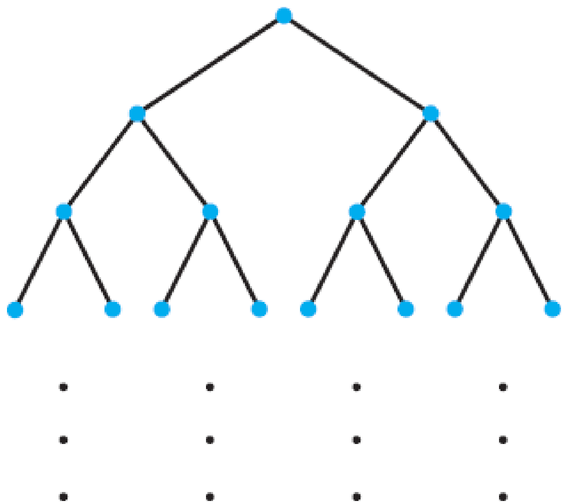


FIGURE 15-8 Binary trees of height 3

The Maximum and Minimum Heights of a Binary Tree



Level	Number of nodes at this level	Total number of nodes at this level and all previous levels
1	$1 = 2^0$	$1 = 2^1 - 1$
2	$2 = 2^1$	$3 = 2^2 - 1$
3	$4 = 2^2$	$7 = 2^3 - 1$
4	$8 = 2^3$	$15 = 2^4 - 1$
⋮	⋮	⋮
h	2^{h-1}	$2^h - 1$

FIGURE 15-9 Counting the nodes in a full binary tree of height h

Facts about Full Binary Trees

- A full binary tree of height $h \geq 0$ has $2^h - 1$ nodes.
- You cannot add nodes to a full binary tree without increasing its height.
- The maximum number of nodes that a binary tree of height h can have is $2^h - 1$.
- The minimum height of a binary tree with n nodes is $[\log_2 (n + 1)]$

The Maximum and Minimum Heights of a Binary Tree

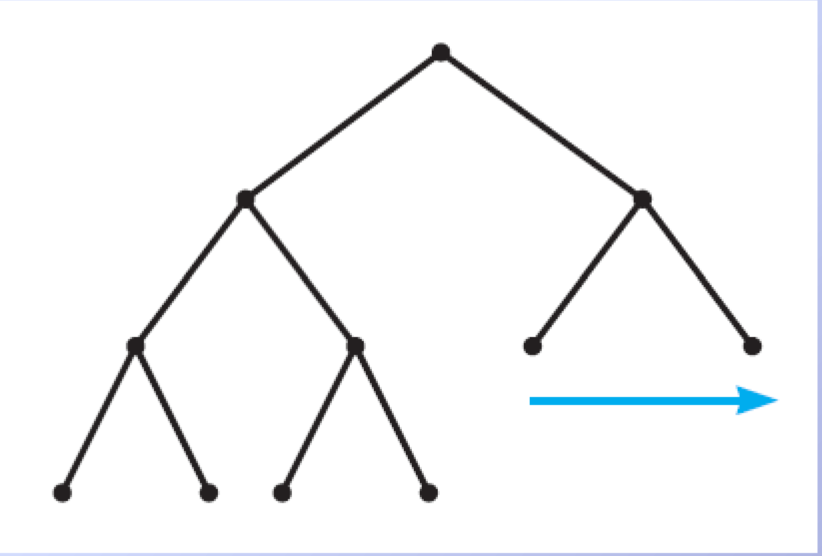


FIGURE 15-10 Filling in the last level of a tree

Traversals of a Binary Tree

- General form of recursive transversal algorithm

```
if (T is not empty)
{
   Display the data in T's root
   Traverse T's left subtree
   Traverse T's right subtree
}
```

Traversals of a Binary Tree

- Preorder traversal.

```
// Traverses the given binary tree in preorder.
// Assumes that "visit a node" means to process the node's data item.
preorder(binTree: BinaryTree): void

   if (binTree is not empty)
   {
      Visit the root of binTree
      preorder(Left subtree of binTree's root)
      preorder(Right subtree of binTree's root)
   }
```

Traversals of a Binary Tree

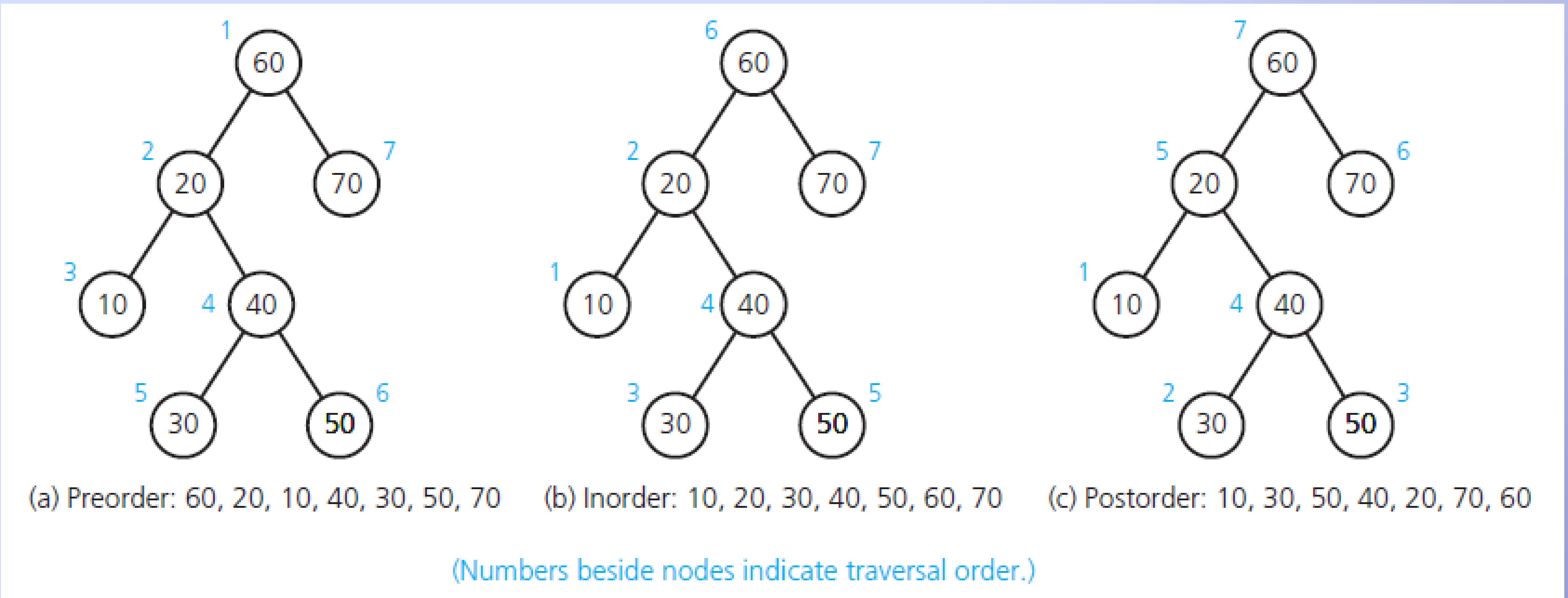


FIGURE 15-11 Three traversals of a binary tree

Traversals of a Binary Tree

- Inorder traversal.

```
// Traverses the given binary tree in inorder.
// Assumes that "visit a node" means to process the node's data item.
inorder(binTree: BinaryTree): void

   if (binTree is not empty)
   {
      inorder(Left subtree of binTree's root)
      Visit the root of binTree
      inorder(Right subtree of binTree's root)
   }
```

Traversals of a Binary Tree

- Postorder traversal.

```
// Traverses the given binary tree in postorder.
// Assumes that "visit a node" means to process the node's data item.
postorder(binTree: BinaryTree): void

   if (binTree is not empty)
   {
      postorder(Left subtree of binTree's root)
      postorder(Right subtree of binTree's root)
      Visit the root of binTree
   }
```

Binary Tree Operations

- Test whether a binary tree is empty.
- Get the height of a binary tree.
- Get the number of nodes in a binary tree.
- Get the data in a binary tree's root.
- Set the data in a binary tree's root.
- Add a new node containing a given data item to a binary tree.

Binary Tree Operations

- Remove the node containing a given data item from a binary tree.
- Remove all nodes from a binary tree.
- Retrieve a specific entry in a binary tree.
- Test whether a binary tree contains a specific entry.
- Traverse the nodes in a binary tree in preorder, inorder, or postorder.

Binary Tree Operations

```
BinaryTree
-----------------------------------------------------------

-----------------------------------------------------------
+isEmpty(): boolean
+getHeight(): integer
+getNumberOfNodes(): integer
+getRootData(): ItemType
+setRootData(newData: ItemType): void
+add(newData: ItemType): boolean
+remove(data: ItemType): boolean
+clear(): void
+getEntry(anEntry: ItemType): ItemType
+contains(data: ItemType): boolean
+preorderTraverse(visit(item: ItemType): void): void
+inorderTraverse(visit(item: ItemType): void): void
+postorderTraverse(visit(item: ItemType): void): void
```

FIGURE 15-12 UML diagram for the class BinaryTree

Binary Tree Operations

- Formalized specifications demonstrated in interface template for binary tree
  - Listing 15-1
- Note method names matching UML diagram
  - Figure 15-12

.htm code listing files must be in the same folder as the .ppt files for these links to work

The ADT Binary Search Tree

- ADT binary tree ill suited for search for specific item
- Binary *search* tree solves problem
- Properties of each node, *n*
  - *n* ’s value greater than all values in left subtree T_L
  - *n* ’s value less than all values in right subtree T_R
  - Both T_R and T_L are binary search trees.

The ADT Binary Search Tree

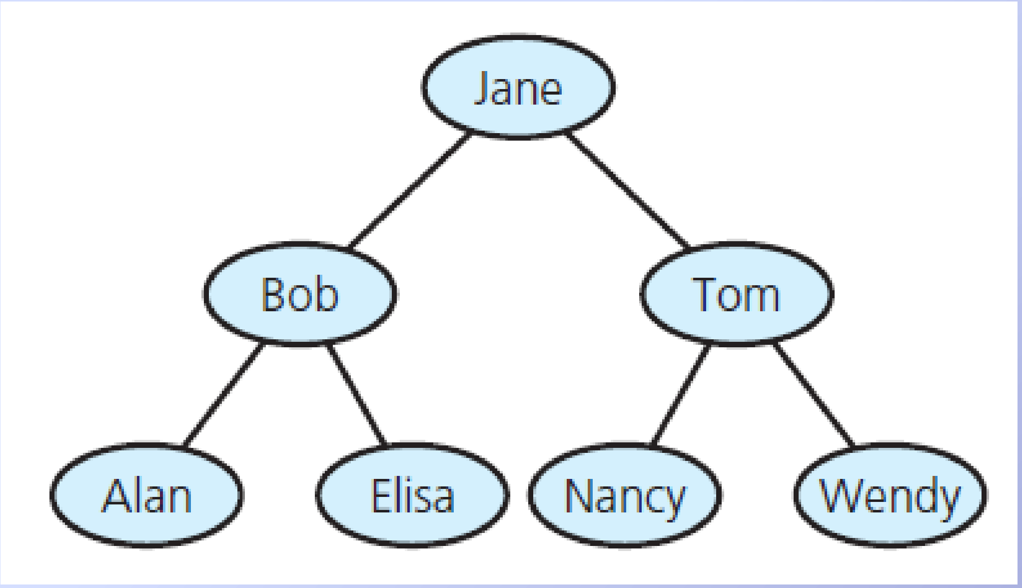


FIGURE 15-13 A binary search tree of names

The ADT Binary Search Tree

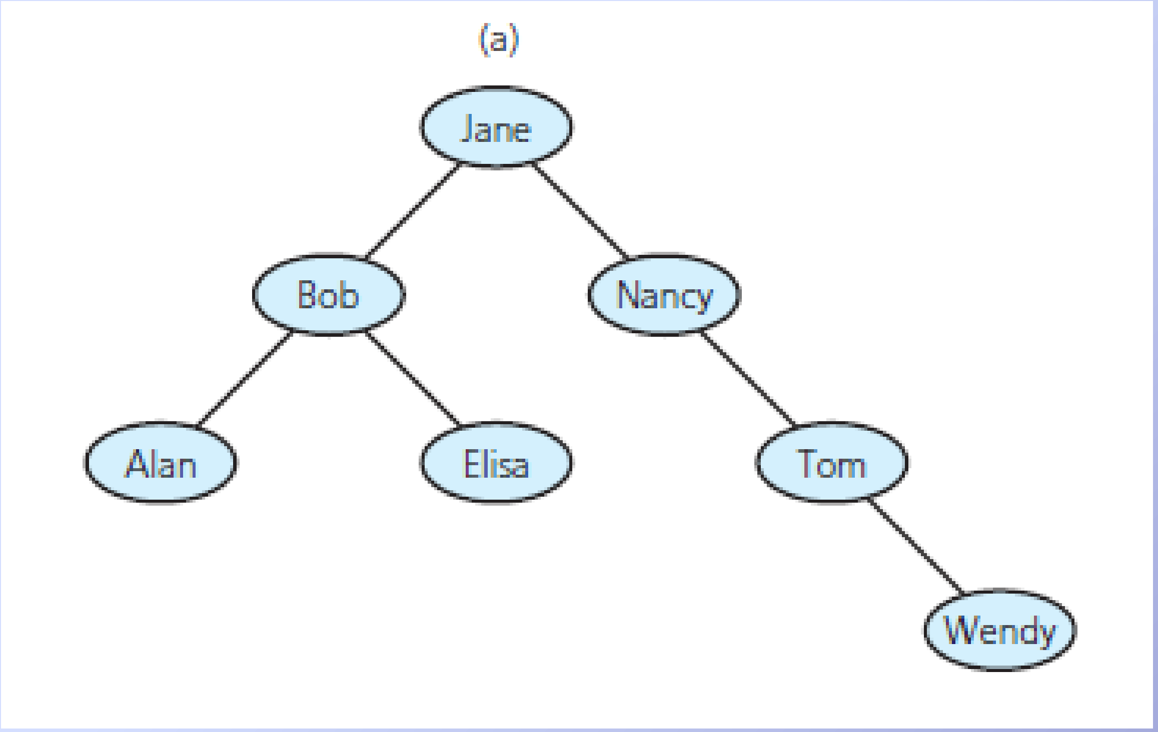


FIGURE 15-14 Binary search trees with the same data as in Figure 15-13

The ADT Binary Search Tree

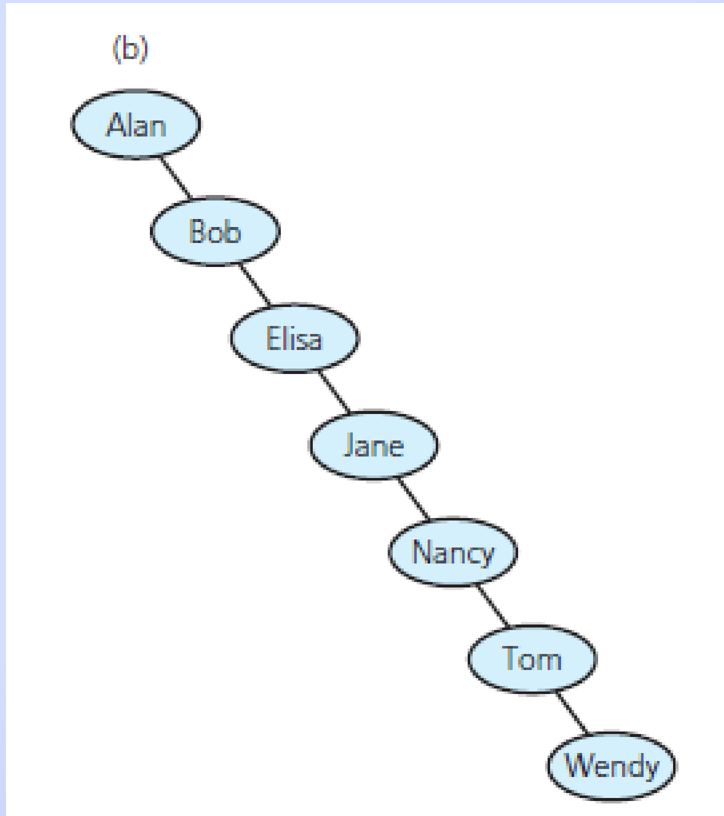


FIGURE 15-14 Binary search trees with the same data as in Figure 15-13

The ADT Binary Search Tree

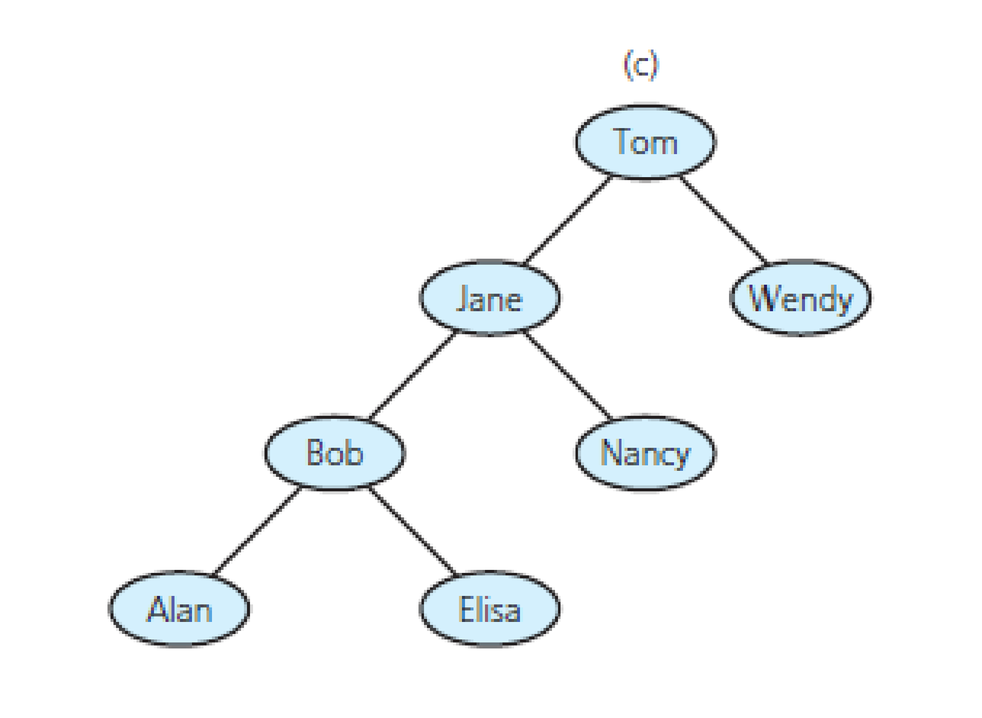


FIGURE 15-14 Binary search trees with the same data as in Figure 15-13

Binary Search Tree Operations

- Test whether binary search tree is empty.
- Get height of binary search tree.
- Get number of nodes in binary search tree.
- Get data in binary search tree's root.
- Insert new item into binary search tree.
- Remove given item from binary search tree.

Binary Search Tree Operations

- Remove all entries from binary search tree.
- Retrieve given item from binary search tree.
- Test whether binary search tree contains specific entry.
- Traverse items in binary search tree in
  - Preorder
  - Inorder
  - Postorder.

Searching a Binary Search Tree

- Search algorithm for binary search tree

```
// Searches the binary search tree for a given target value.
search(bstTree: BinarySearchTree, target: ItemType)

   if (bstTree is empty)
      The desired item is not found
   else if (target == data item in the root of bstTree)
      The desired item is found
   else if (target < data item in the root of bstTree)
      search(Left subtree of bstTree, target)
   else
      search(Right subtree of bstTree, target)
```

Searching a Binary Search Tree

Alan	Bob	Elisa	Jane	Nancy	Tom	Wendy
0	1	2	3	4	5	6

FIGURE 15-15 An array of names in sorted order

Creating a Binary Search Tree

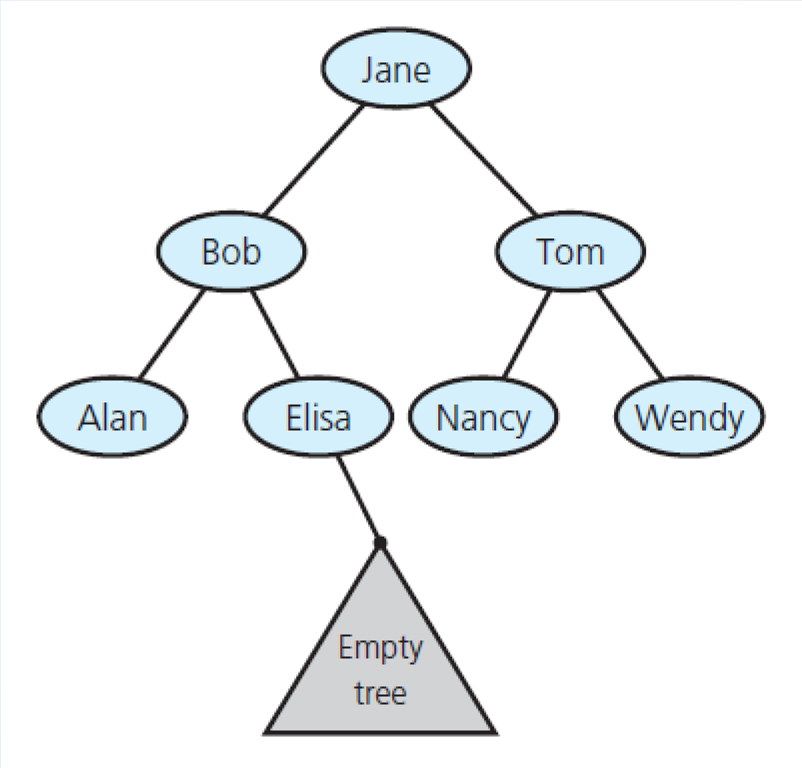


FIGURE 15-16 Empty subtree where the **`search`** algorithm terminates when looking for Frank

Traversals of a Binary Search Tree

- Algorithm

```
// Traverses the given binary tree in inorder.
// Assumes that "visit a node" means to process the node's data item.
inorder(binTree: BinaryTree): void

   if (binTree is not empty)
   {
      inorder(Left subtree of binTree's root)
      Visit the root of binTree
      inorder(Right subtree of binTree's root)
   }
```

Efficiency of Binary Search Tree Operations

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

FIGURE 15-17 The Big O for the retrieval, insertion, removal, and traversal operations of the ADT binary search tree

End

Chapter 15