Tree Implementations

Chapter 16

Contents

- The Nodes in a Binary Tree
- A Link-Based Implementation of the ADT Binary Tree
- A Link-Based Implementation of the ADT Binary Search Tree
- Saving a Binary Search Tree in a File
- Tree Sort
- General Trees

Array-Based Representation

- Consider required data members

```
TreeNode<ItemType>  tree[MAX_NODES];  // Array of tree nodes
int                 root;             // Index of root
int                 free;             // Index of free list
```

- View class **`TreeNode`**,

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

Array-Based Representation

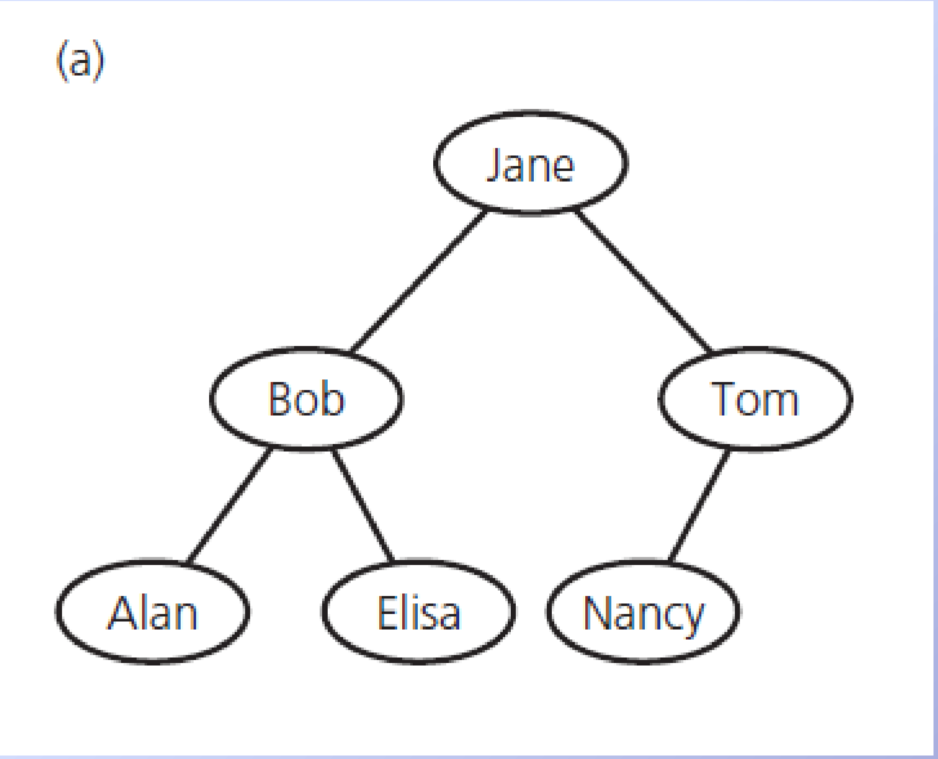


FIGURE 16-1 (a) A binary tree of names;

Array-Based Representation

(b)

	tree item	leftChild	rightChild
0	Jane	1	2
1	Bob	3	4
2	Tom	5	–1
3	Alan	–1	–1
4	Elisa	–1	–1
5	Nancy	–1	–1
6	?	–1	7
7	?	–1	8
8	?	–1	9
.	.	.	.

root: 0

free: 6

Free list

FIGURE 16-1 (b) its implementation using the array **`tree`**

Link-Based Representation

- Listing 16-2 shows the class **`BinaryNode`** for a link-based implementation

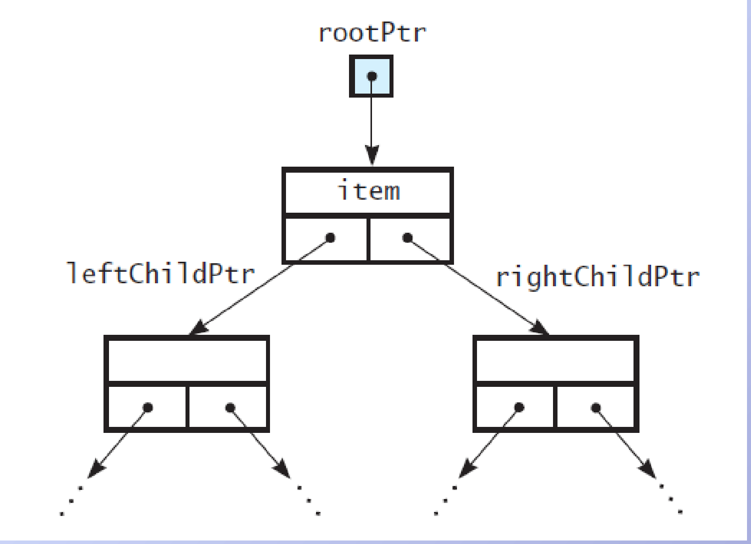


FIGURE 16-2 A link-based implementation of a binary tree

Link-Based Implementation

- View header file for the link-based implementation of the class **`BinaryNodeTree`**, Listing 16-3
- Note significant portions of the implementation file, Listing 16-A

Link-Based Implementation

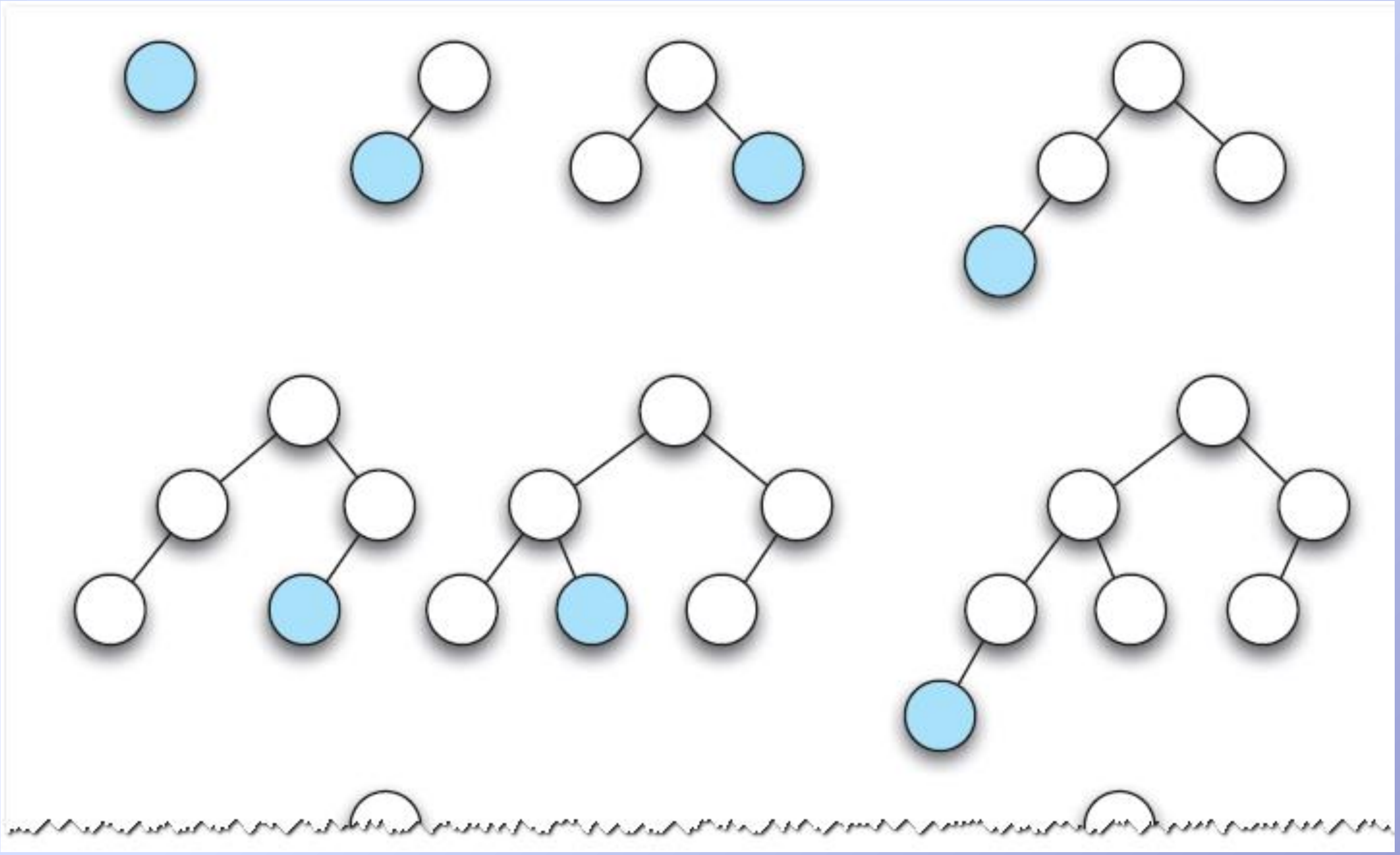


FIGURE 16-3 Adding nodes to an initially empty binary tree

Link-Based Implementation

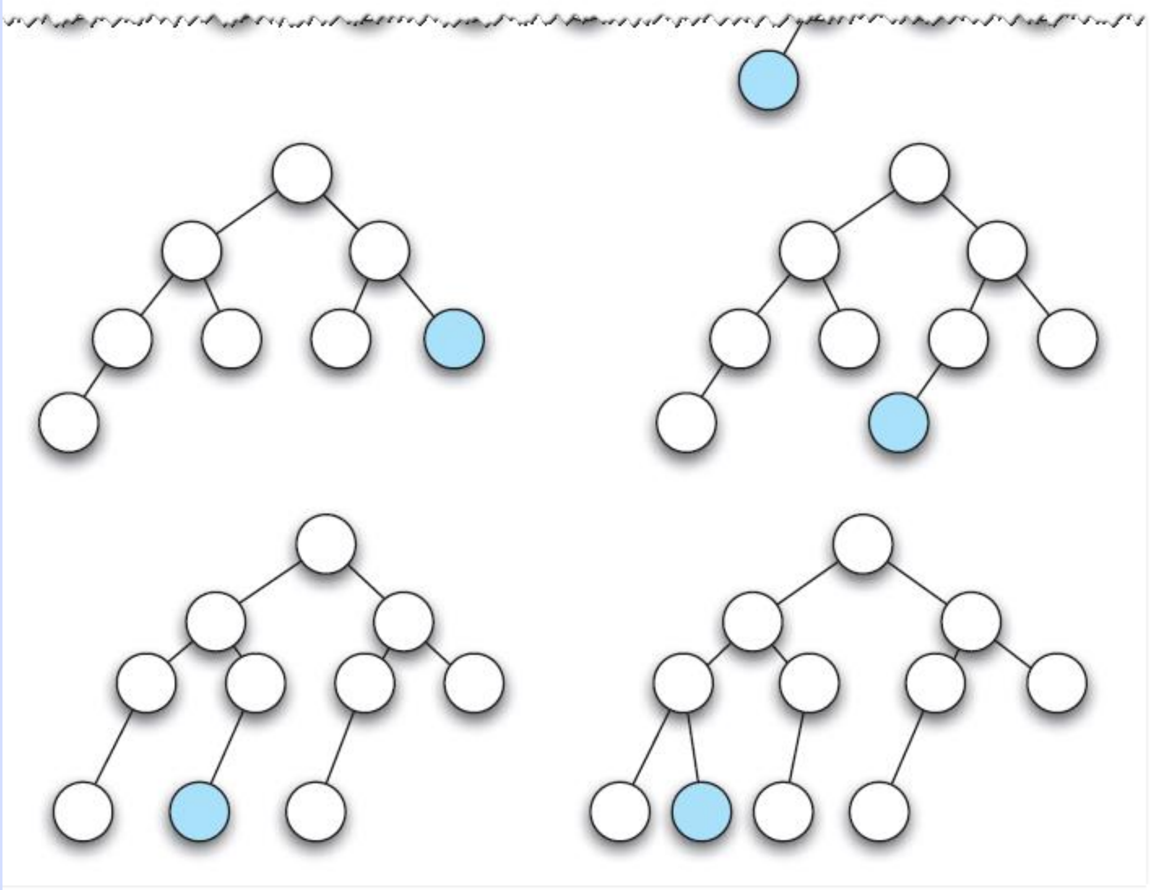


FIGURE 16-3 Adding nodes to an initially empty binary tree

Link-Based Implementation

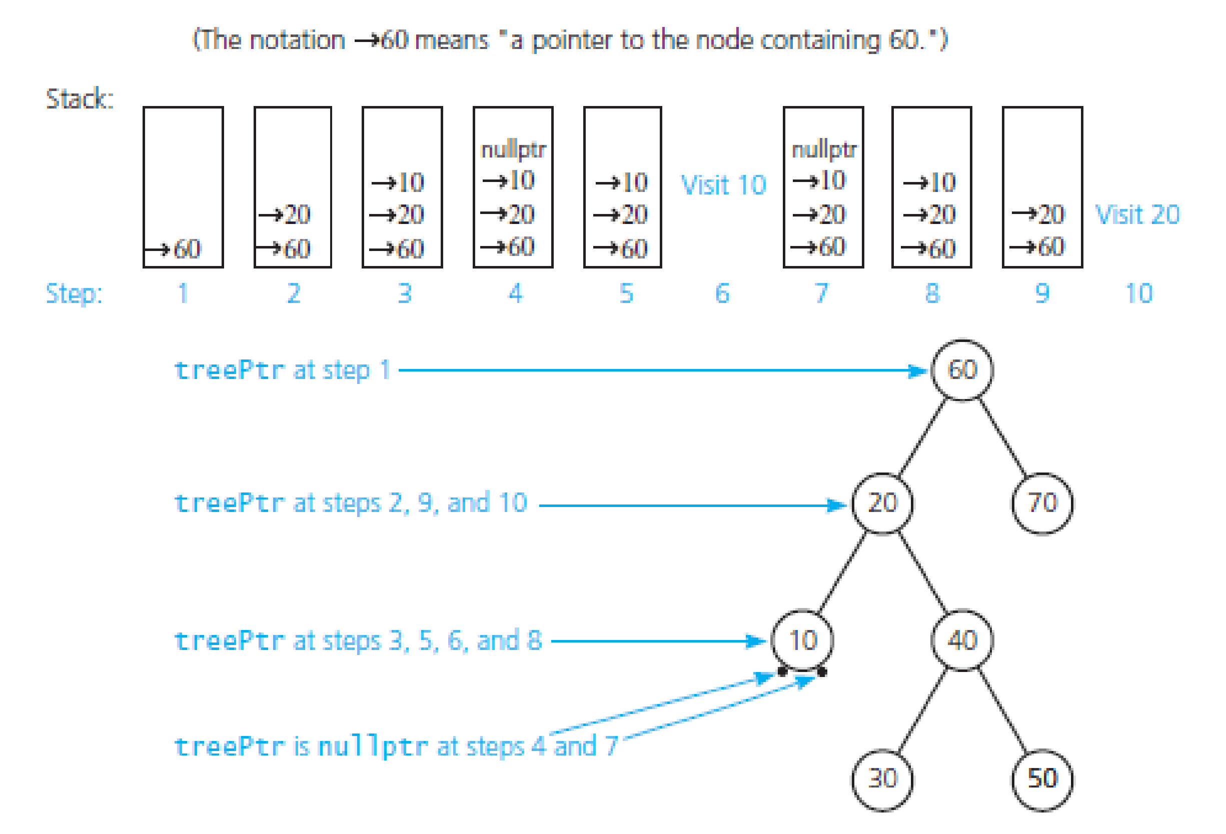


FIGURE 16-4 Contents of the implicit stack as **`treePtr`** progresses through a given tree during a recursive inorder traversal

Link-Based Implementation

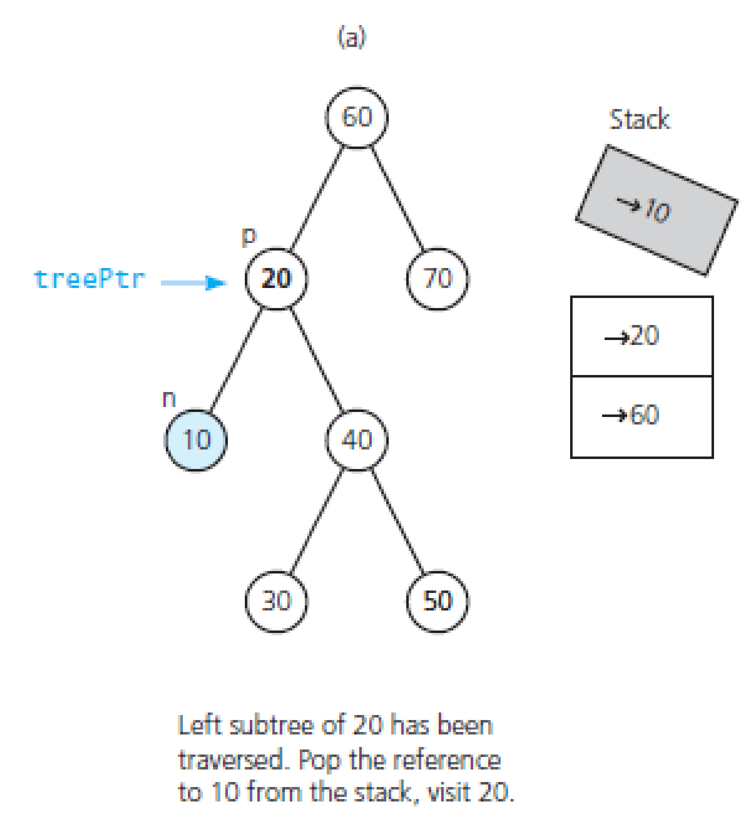


FIGURE 16-5 Traversing (a) the left subtree (steps 9 and 10 in Figure 16-4)

Link-Based Implementation

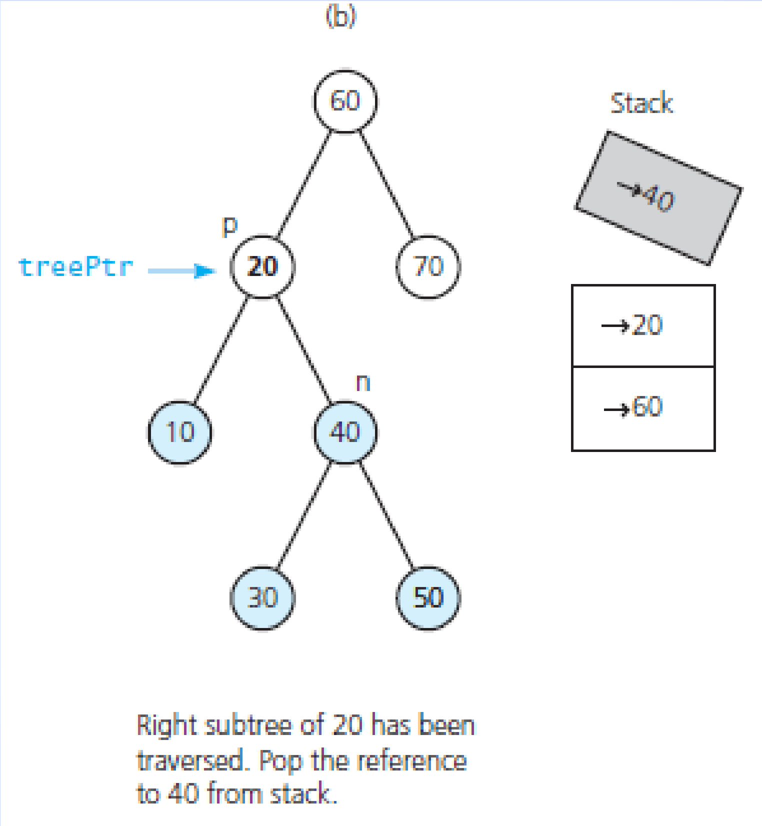
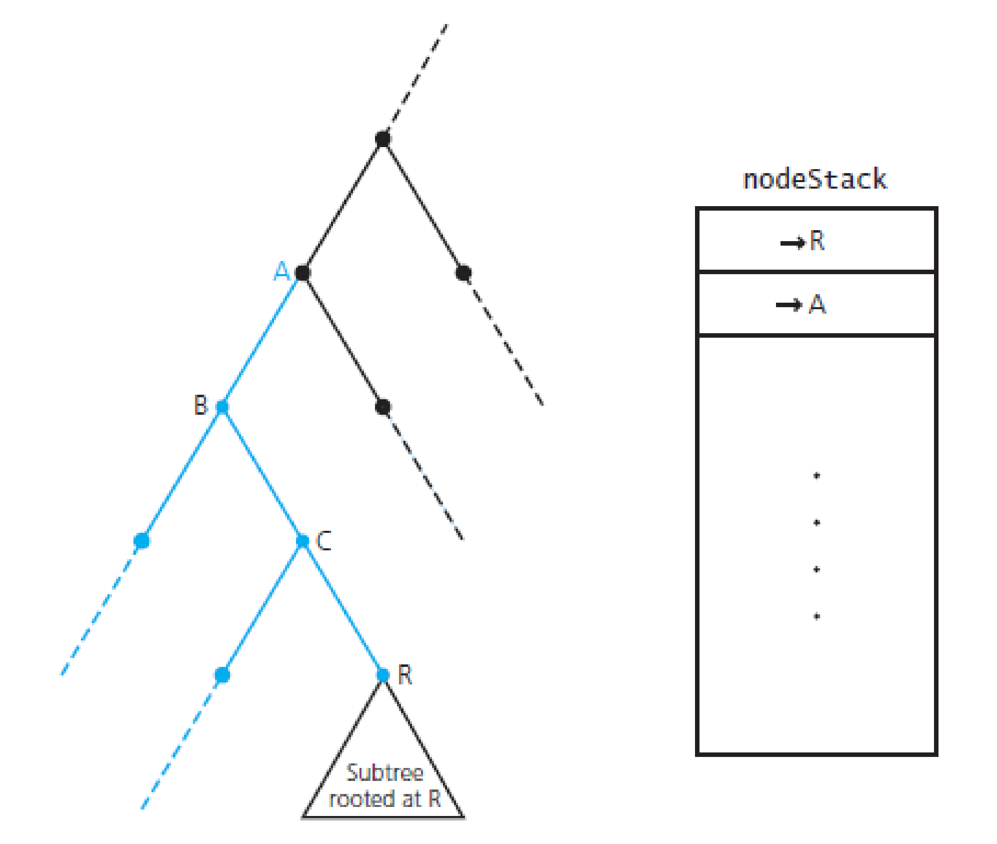


FIGURE 16-5 Traversing (b) the right subtree of 20

Link-Based Implementation



View pseudocode for non recursive traversal, Listing 16-B

FIGURE 16-6 Avoiding returns to nodes B and C

Algorithms for the ADT Binary Search Tree Operations

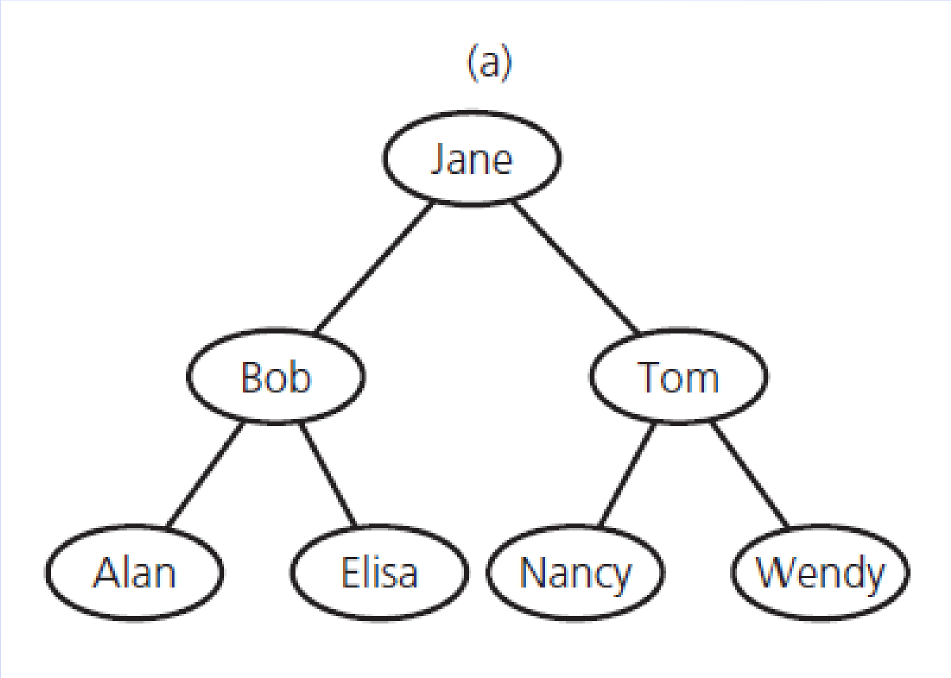


FIGURE 16-7 (a) A binary search tree;

Algorithms for the ADT Binary Search Tree Operations

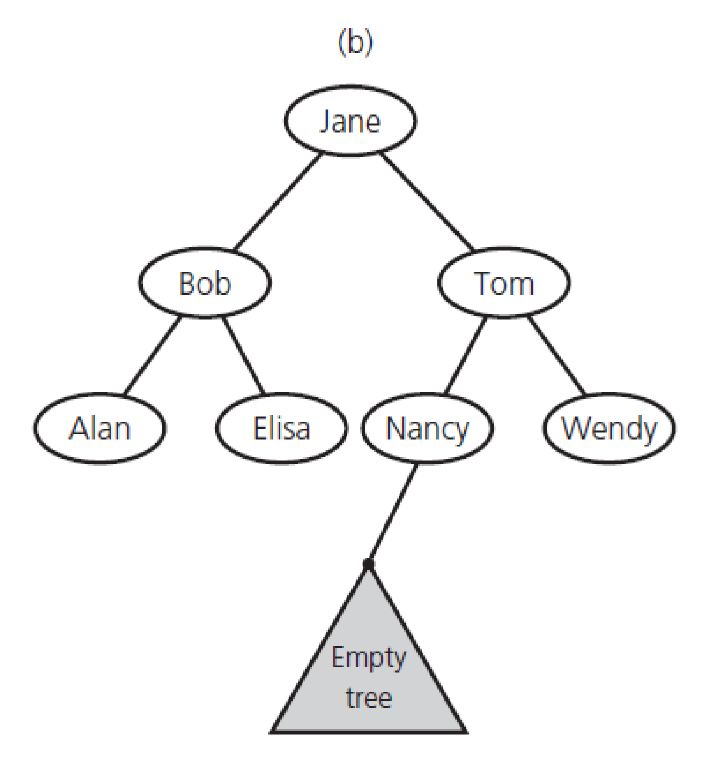


FIGURE 16-7 (b) empty subtree where the `search` algorithm terminates when looking for Kody

Algorithms for the ADT Binary Search Tree Operations

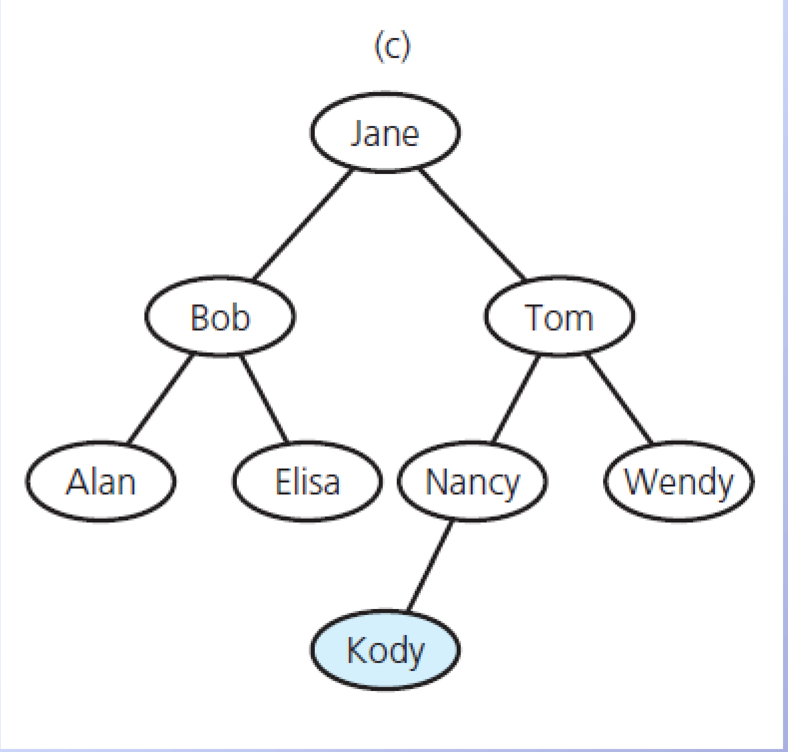


FIGURE (c) the tree after Kody is inserted as a new leaf

Algorithms for the ADT Binary Search Tree Operations

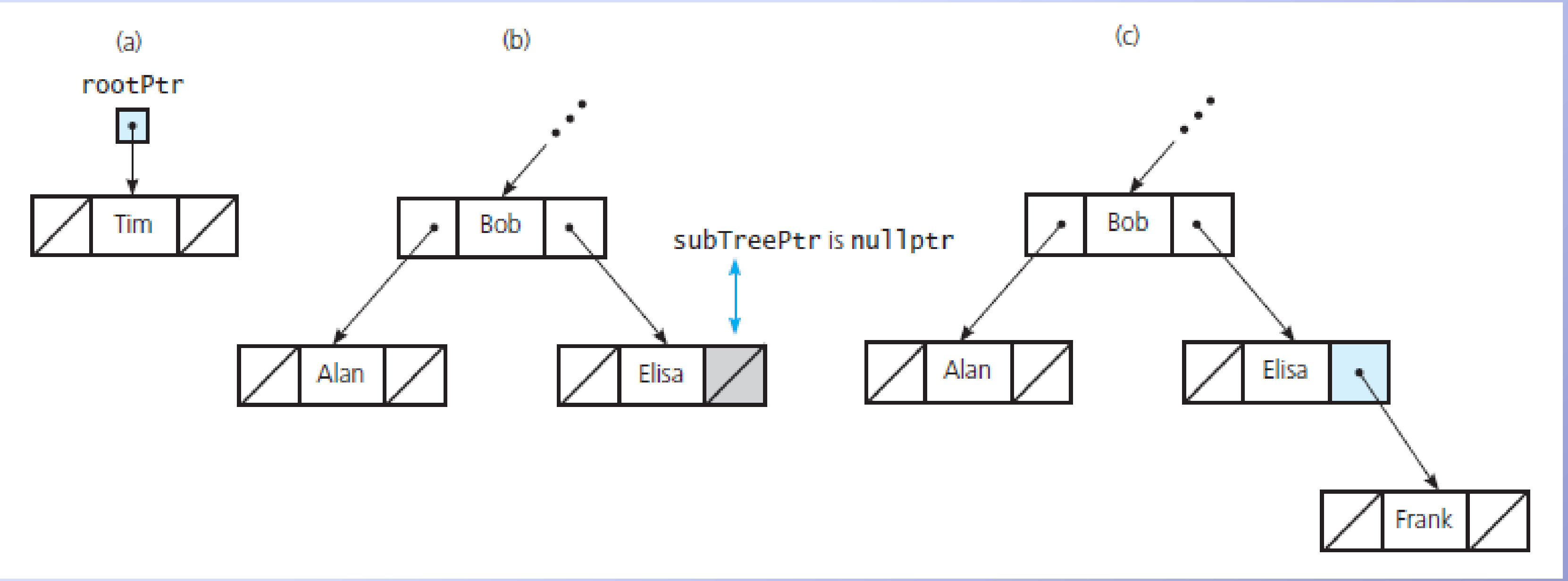


FIGURE 16-8 (a) Insertion into an empty tree; (b) search for Frank terminates at a leaf; (c) insertion at a leaf

Algorithms for the ADT Binary Search Tree Operations

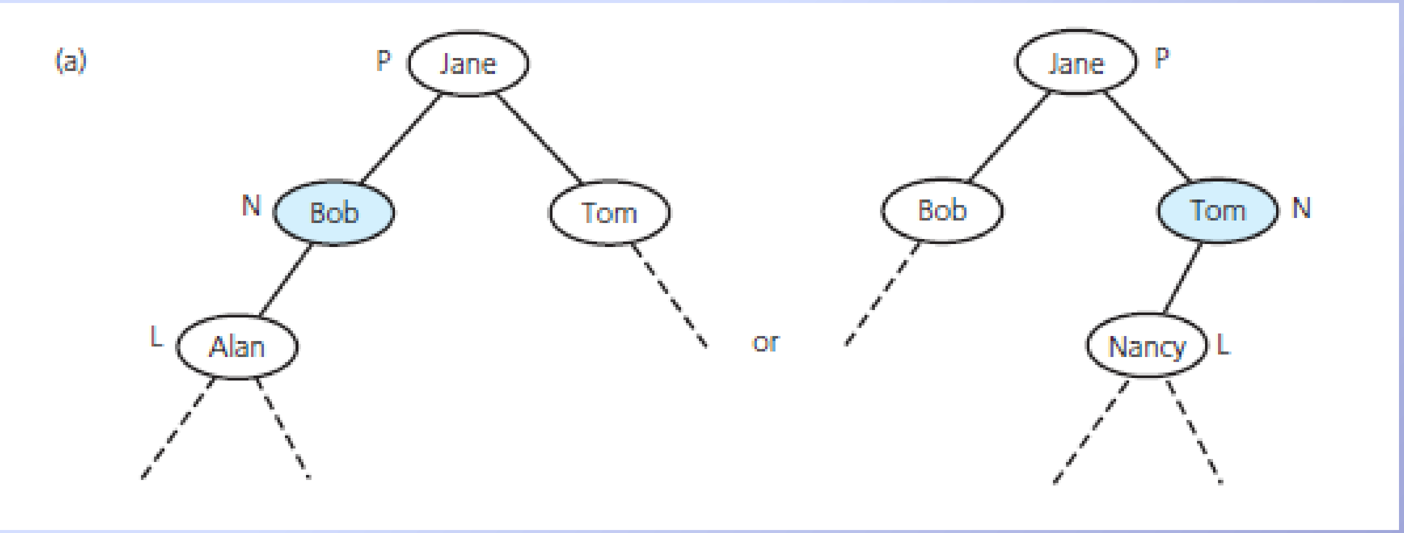


FIGURE 16-9 (a) N with only a left child— N can be either the left child or right child of P ;

Algorithms for the ADT Binary Search Tree Operations

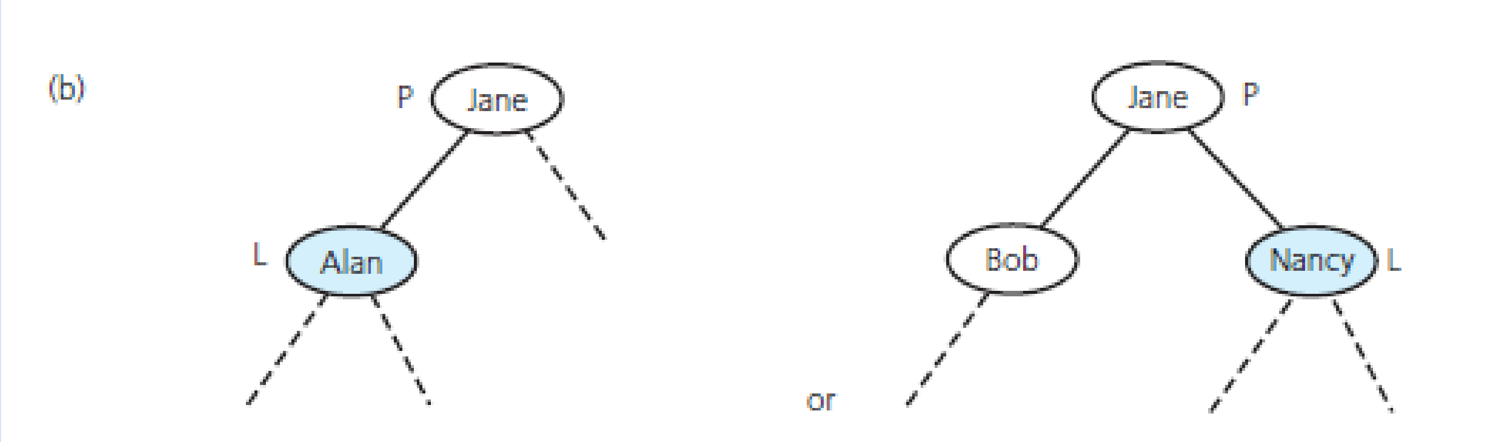


FIGURE 16-9 (b) after removing node N

Algorithms for the ADT Binary Search Tree Operations

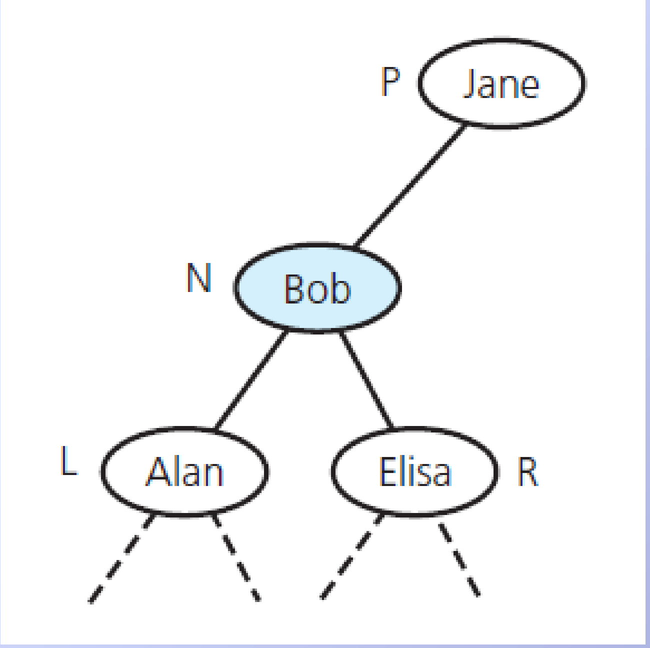


FIGURE 16-10 N with two children

Algorithms for the ADT Binary Search Tree Operations

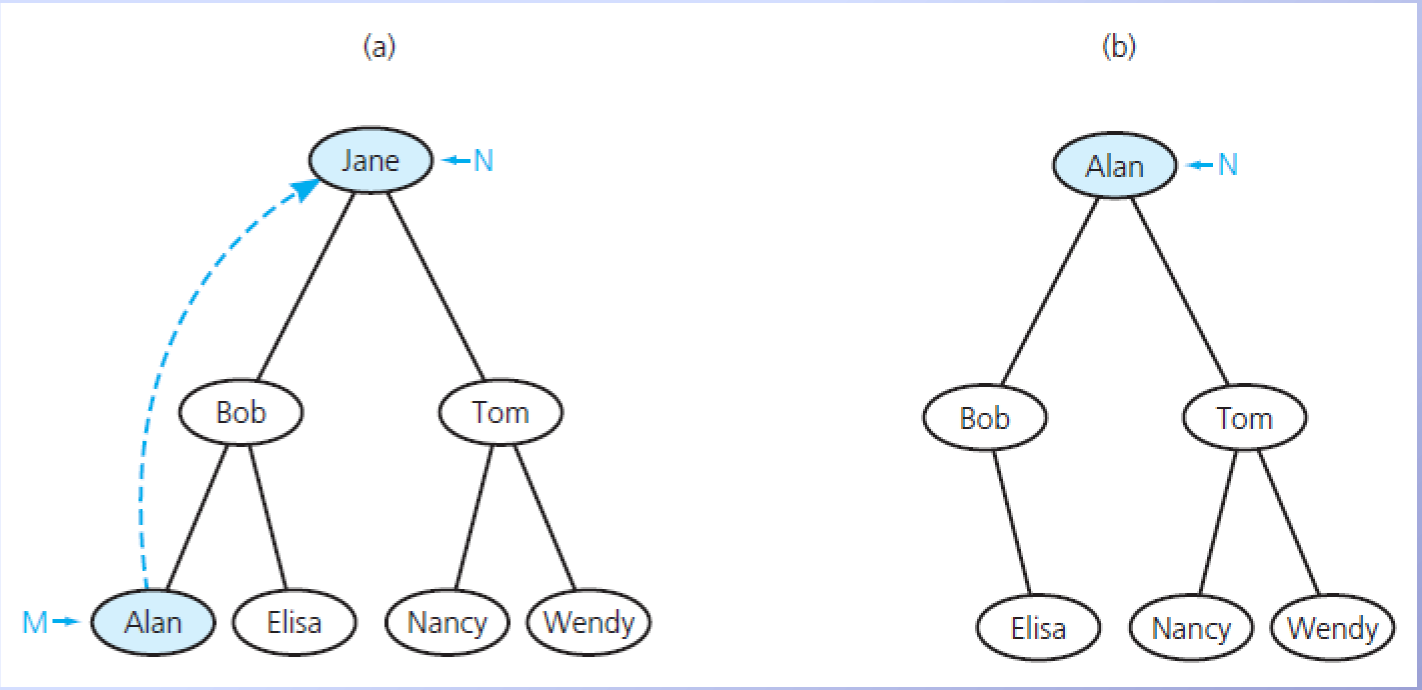


FIGURE 16-11 (a) Not any node will do; (b) no longer a binary search tree

Algorithms for the ADT Binary Search Tree Operations

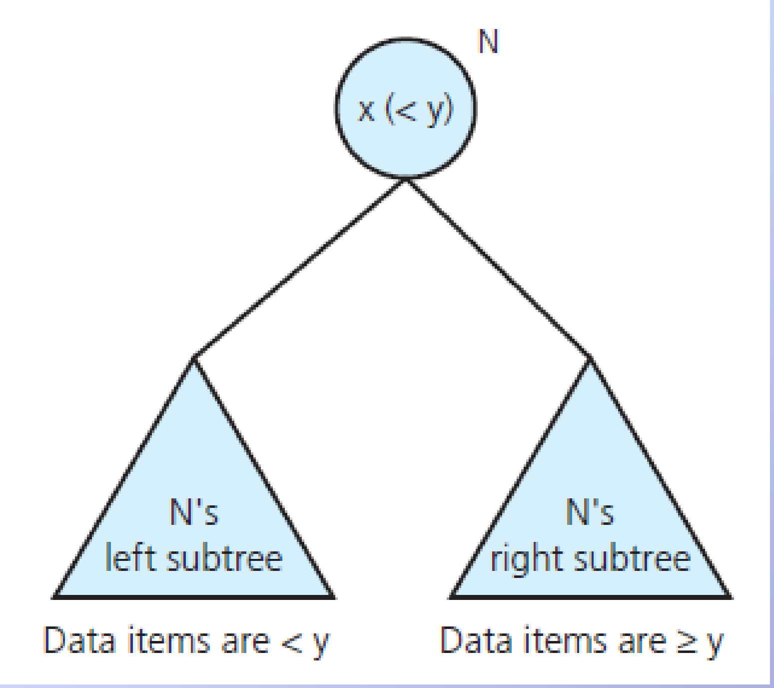
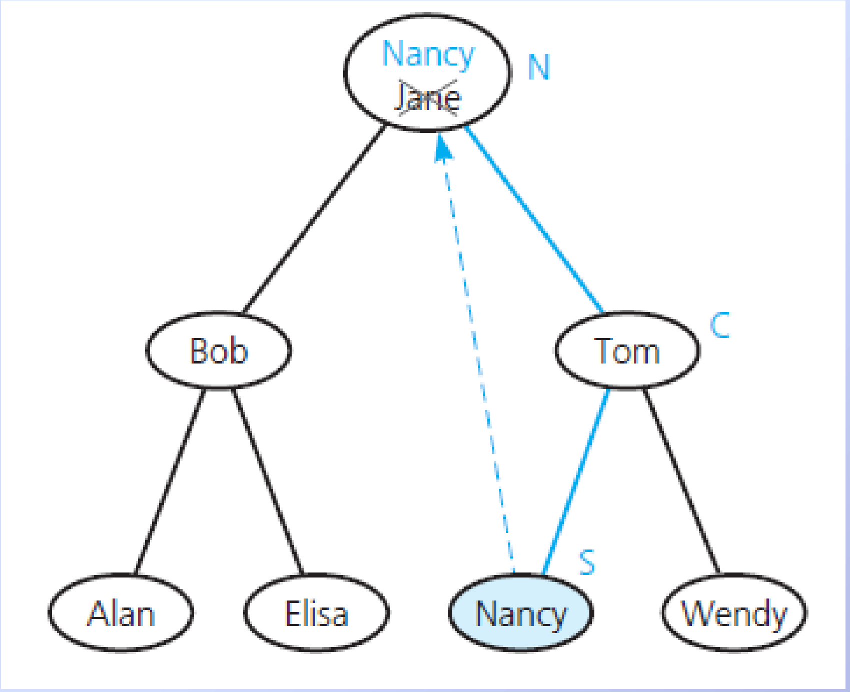


FIGURE 16-12 Search key x can be replaced by y

Algorithms for the ADT Binary Search Tree Operations



View final draft of remove algorithm, Listing 16-C

FIGURE 16-13 Copying the item whose search key is the inorder successor of N 's search key

Algorithms for the ADT Binary Search Tree Operations

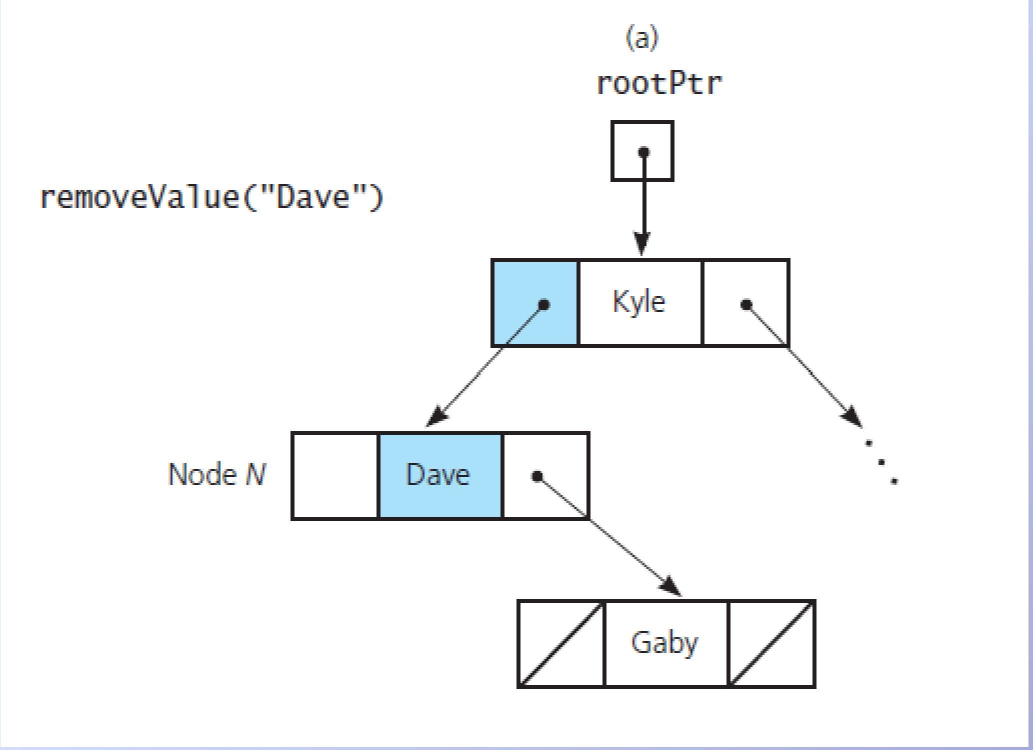


FIGURE 16-14 Recursive deletion of node N

Algorithms for the ADT Binary Search Tree Operations

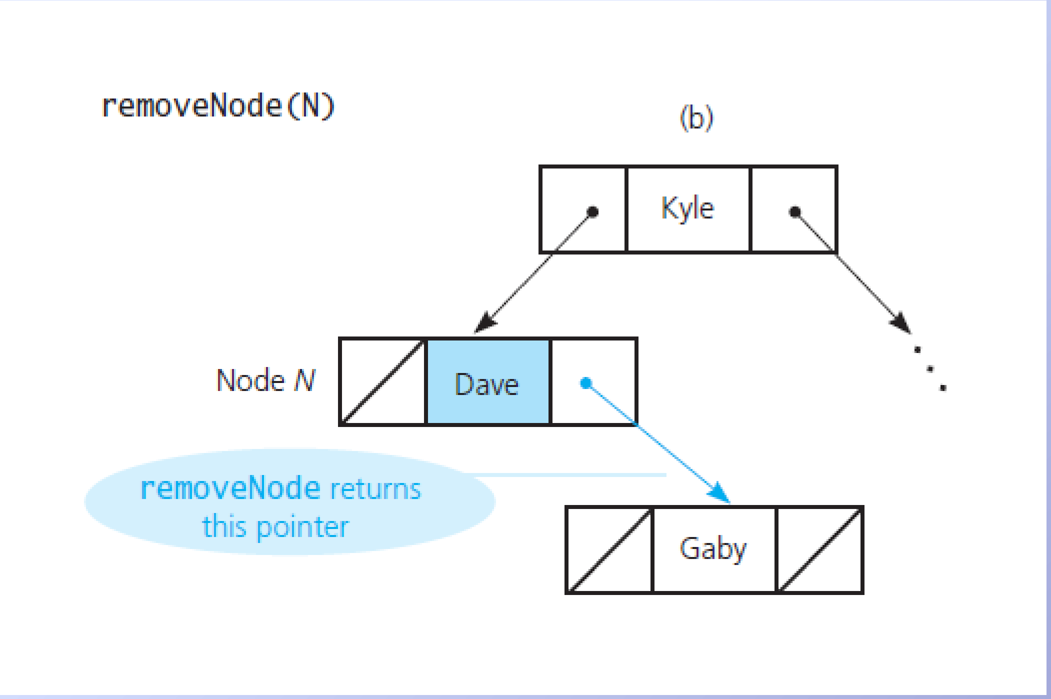


FIGURE 16-14 Recursive deletion of node N

Algorithms for the ADT Binary Search Tree Operations

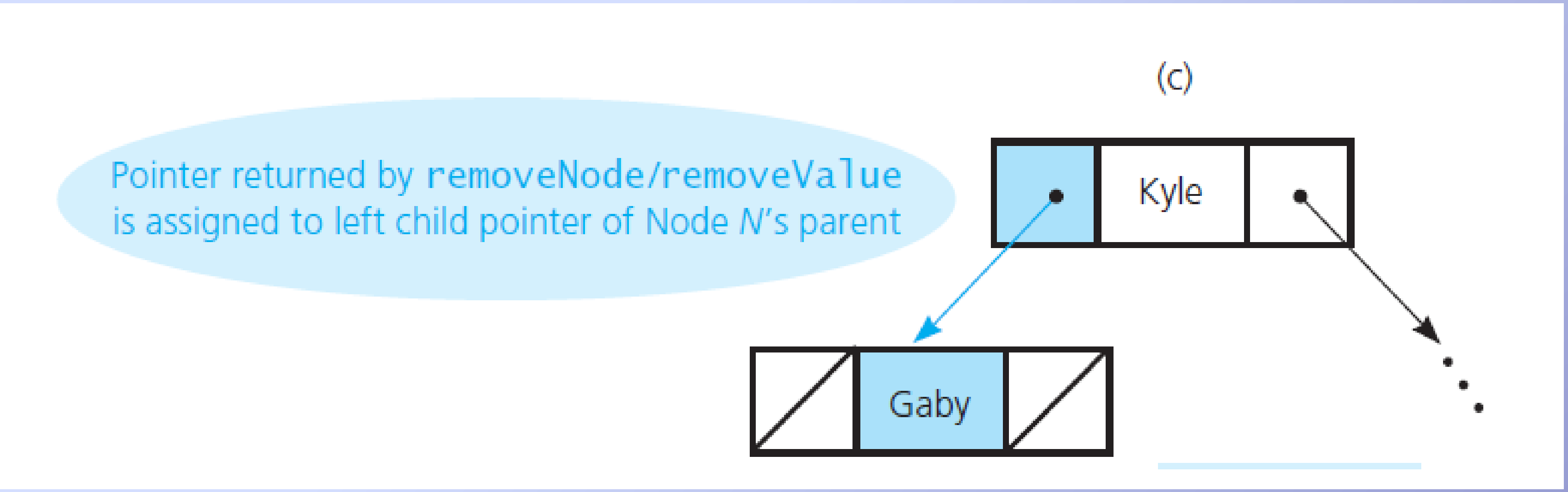


FIGURE 16-14 Recursive deletion of node N

Algorithms for the ADT Binary Search Tree Operations

- **`findNode`** as a refinement of **`search`**

```
// Locates the node in the binary search tree to which subTreePtr points that contains
// the value target. Returns either a pointer to the located node or nullptr if such a
// node is not found.
findNode(subTreePtr: BinaryNodePointer, target: ItemType): BinaryNodePointer

   if (subTreePtr == nullptr)
      return nullptr                              // Not found
   else if (subTreePtr->getItem() == target)
      return subTreePtr;                          // Found
   else if (subTreePtr->getItem() > target)
      // Search left subtree
      return findNode(subTreePtr->getLeftChildPtr(), target)
   else
      // Search right subtree
      return findNode(subTreePtr->getRightChildPtr(), target)
```

The Class **`BinarySearchTree`**

- View header file for the link-based implementation of the class **`BinarySearchTree`**, [Listing 16-4]

Saving Binary Search Tree in a File

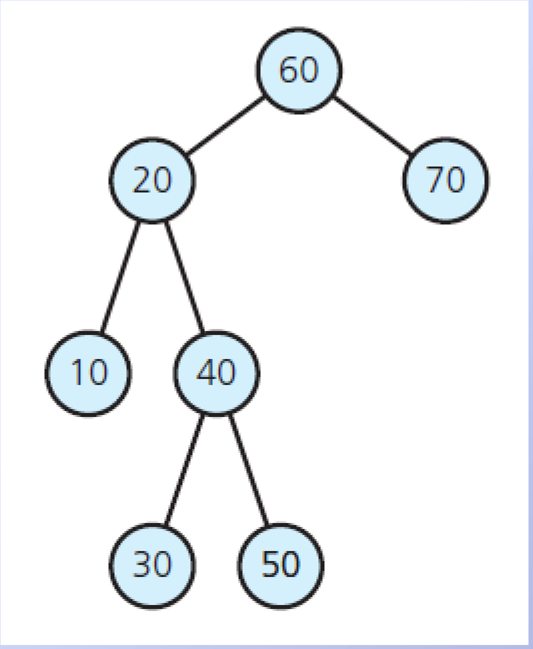


FIGURE 16-15 An initially empty binary search tree after the insertion of 60, 20, 10, 40, 30, 50, and 70

Saving Binary Search Tree in a File

- Recursive algorithm to create full binary search tree with n nodes

```
// Builds a full binary search tree from n sorted values in a file.
// Returns a pointer to the tree's root.
readFullTree(n: integer): BinaryNodePointer

   if (n > 0)
   {
      // Get the root
      treePtr = pointer to new node with nullptr as its child pointers
      rootItem = next item from file
      treePtr->setItem(rootItem)

     // Construct the left subtree
     leftPtr = readFullTree(treePtr->getLeftChildPtr(), n / 2)
     treePtr->setLeftChildPtr(leftPtr)

     // Construct the right subtree
     rightPtr = readFullTree(treePtr->getRightChildPtr(), n / 2)
     treePtr->setRightChildPtr(rightPtr )

     return treePtr
  }
  else
     return nullptr
```

Saving Binary Search Tree in a File

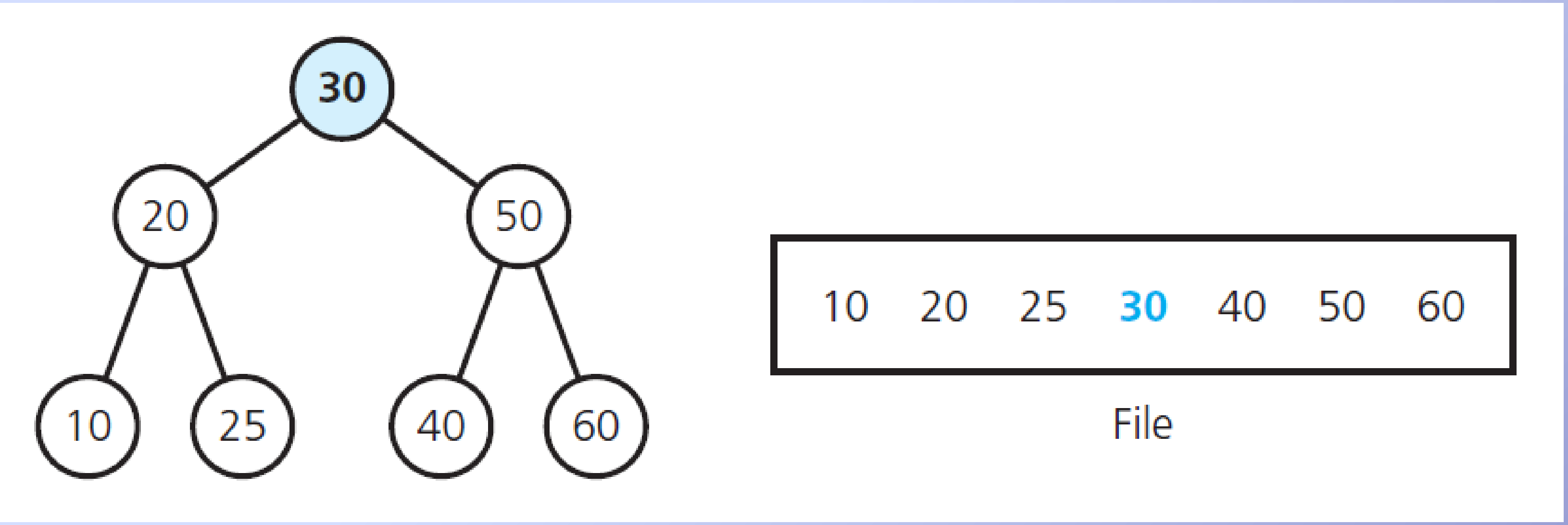


FIGURE 16-16 A full tree saved in a file by using inorder traversal

Saving Binary Search Tree in a File

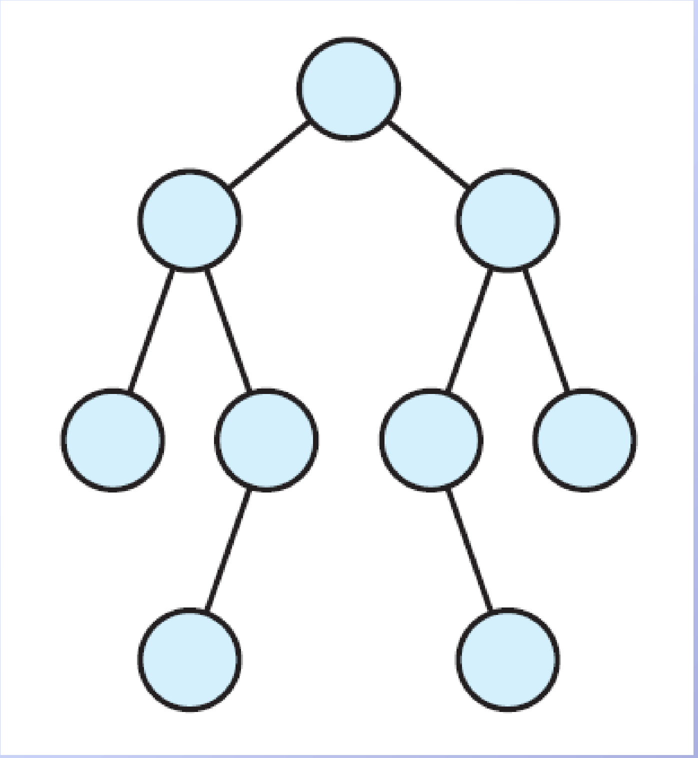


FIGURE 16-17 A tree of minimum height that is not complete

Saving Binary Search Tree in a File

- Building a minimum height binary search tree

```
// Builds a minimum-height binary search tree from n sorted values in a file.
// Returns a pointer to the tree's root.
readTree(n: integer): BinaryNodePointer

   if (n > 0)
   {
      // Get the root
      treePtr = pointer to new node with nullptr as its child pointers
      rootItem = next item from file
      treePtr->setItem(rootItem)

      // Construct the left subtree
      leftPtr = readFullTree(treePtr->getLeftChildPtr(), n / 2)
      treePtr->setLeftChildPtr(leftPtr)

      // Construct the right subtree
      rightPtr = readFullTree(treePtr->getRightChildPtr(), (n - 1) / 2)
      treePtr->setRightChildPtr(rightPtr )

      return treePtr
   }
   else
      return nullptr
```

Tree Sort

- Algorithm for tree sort

```
// Sorts the integers in an array into ascending order.
treeSort(anArray: array, n: integer)

    Insert anArray's entries into a binary search tree bst
    Traverse bst in inorder. As you visit bst's nodes, copy their data items into successive
     locations of anArray
```

General Trees

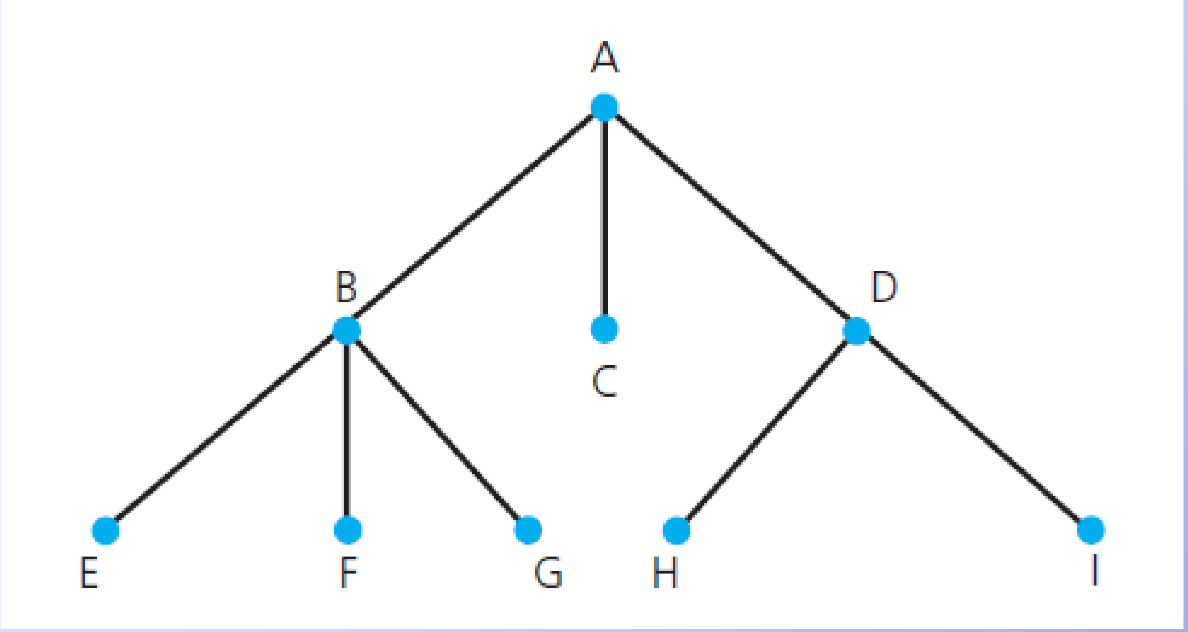


FIGURE 16-18 A general tree

General Trees

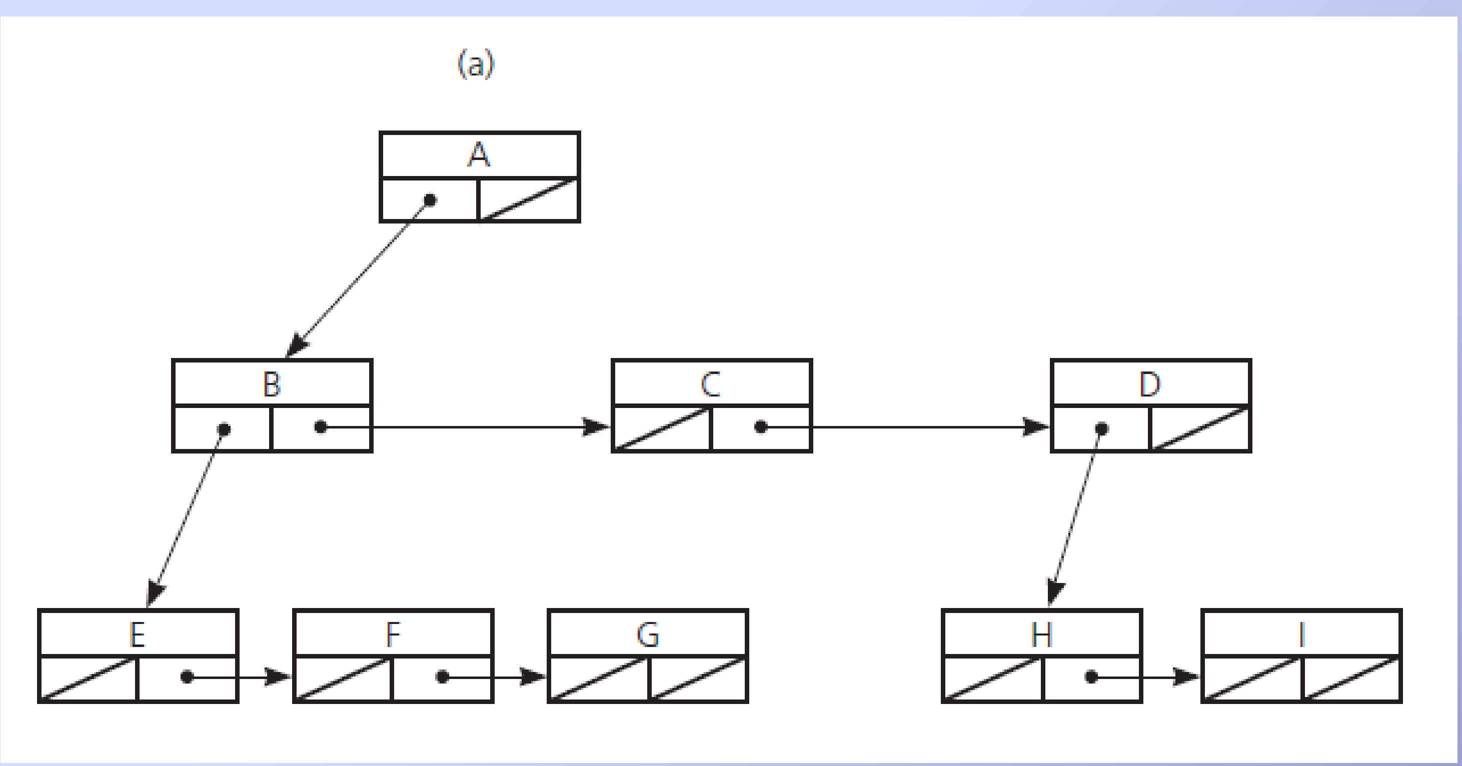


FIGURE 16-19 (a) A link-based implementation of the general tree in Figure 16-18 ;

General Trees

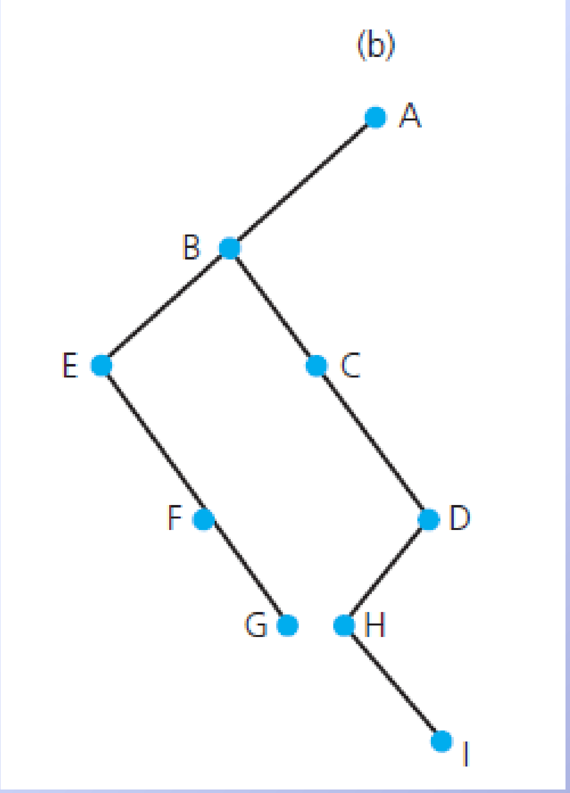


FIGURE 16-19 (b) the binary tree that part *a* represents

General Trees

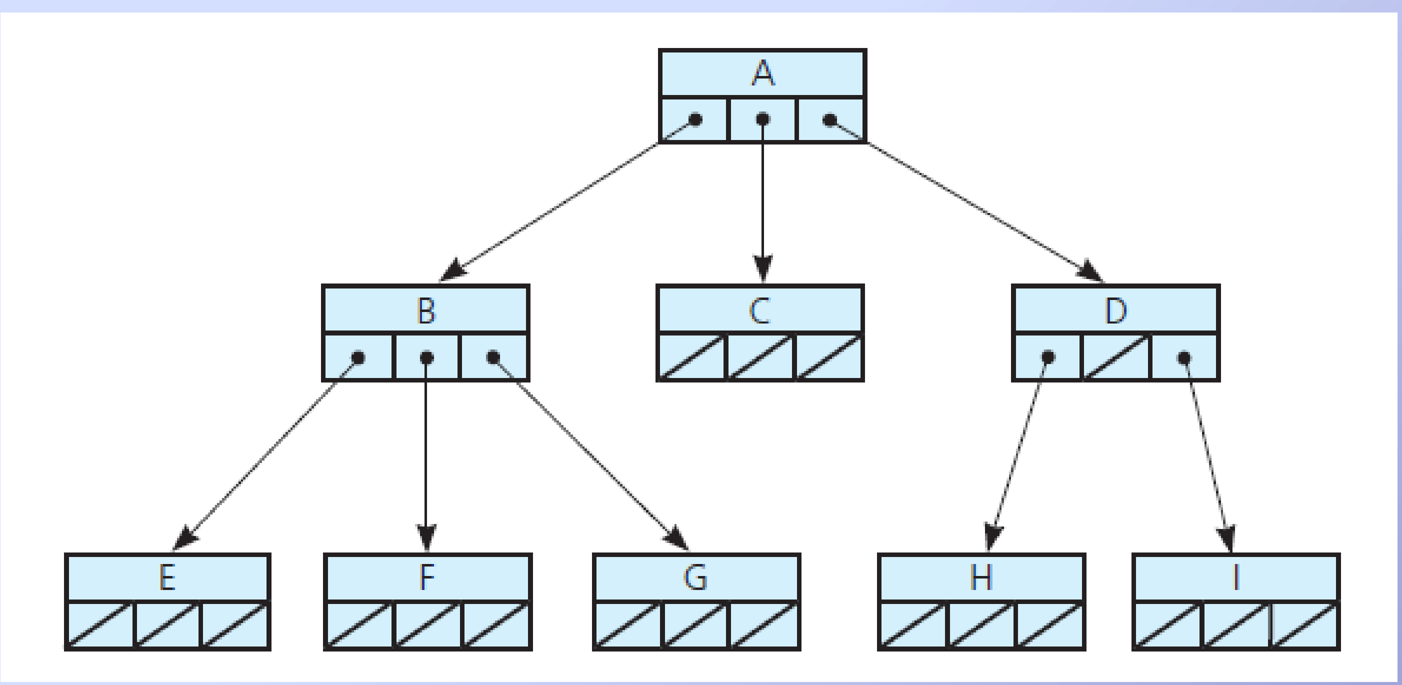


FIGURE 16-20 An implementation of the n -ary tree in Figure 16-18

End

Chapter 16