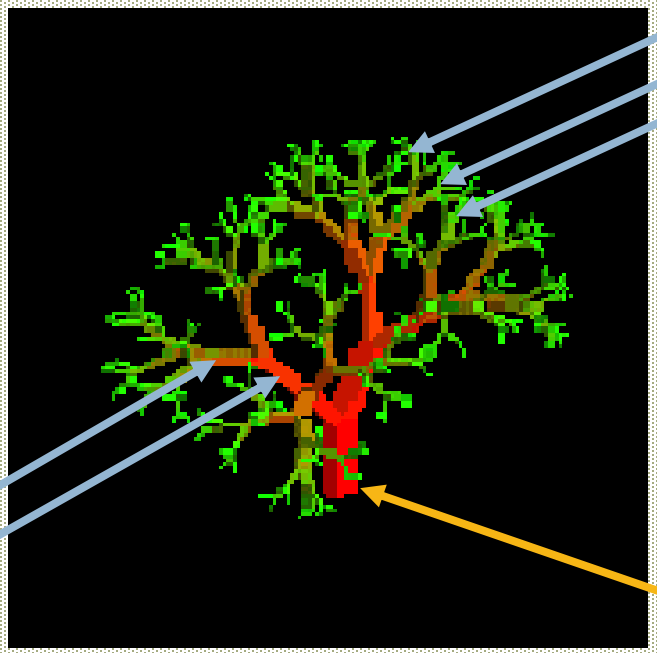
TREES & BINARY TREES

10/24/2018

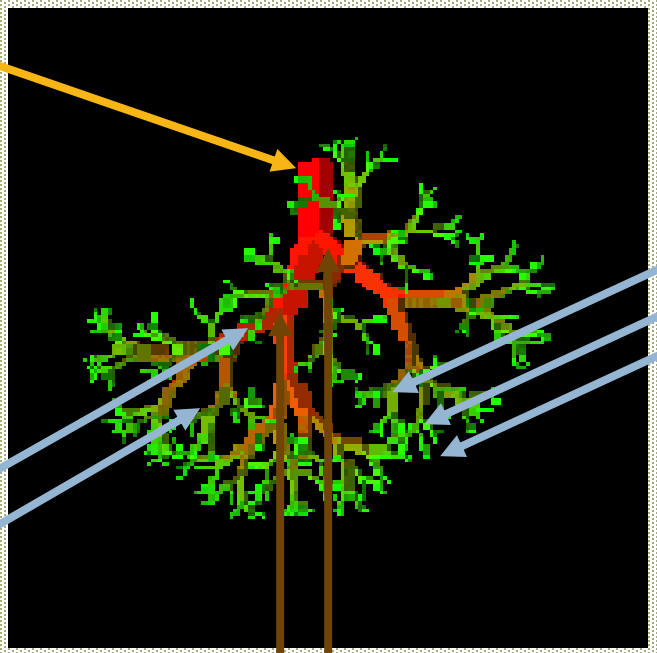
Outline

- Introduction
  - Representation Of Trees
- Binary Trees
- Binary Tree Traversals
- Additional Binary Tree Operations
- Binary Search Trees
- Euler Tours

Nature View of a Tree



Computer Scientist's View



What is a Tree

- A ***tree*** structure means that the data are organized so that items of information are related by branches
- Examples:

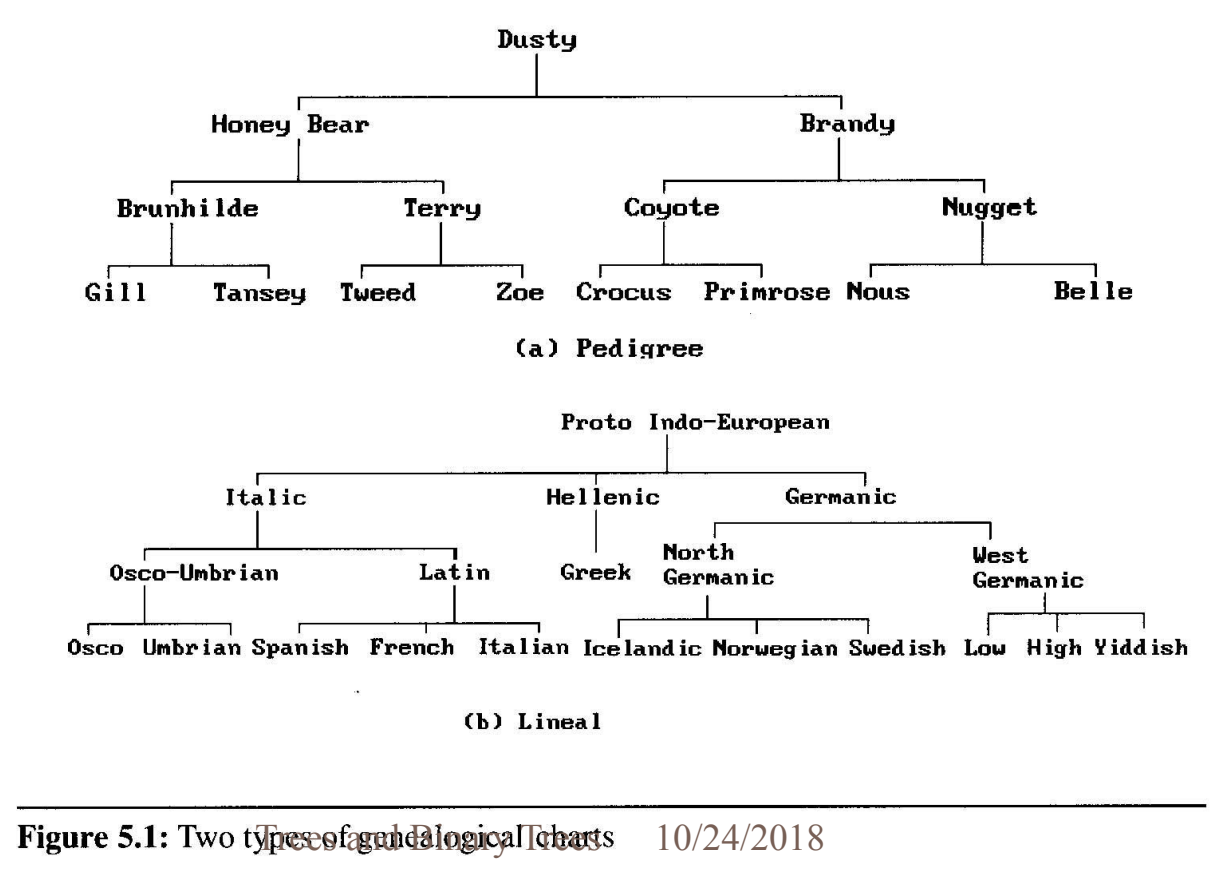
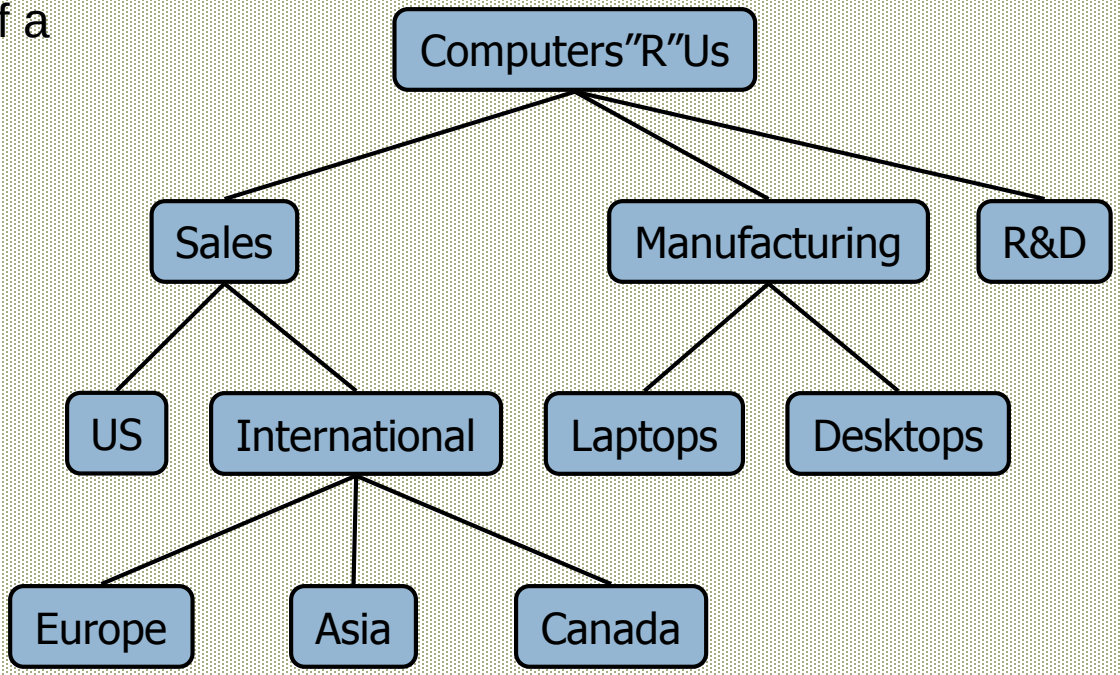


Figure 5.1: Two types of genealogical charts

What is a Tree

- A tree is a finite nonempty set of elements.
- It is an abstract model of a hierarchical structure.
- consists of nodes with a parent-child relation.
- Applications:
  - Organization charts
  - File systems
  - Programming environments



What is a Tree

- **Definition** (recursively): A *tree* is a finite set of one or more nodes such that
  - There is a specially designated node called *root*.
  - The remaining nodes are partitioned into n>=0 disjoint set $T_1,\ldots,T_n$, where each of these sets is a tree. $T_1,\ldots,T_n$ are called the *subtrees* of the root.
- Every node in the tree is the root of some subtree

Tree Terminology

- *Root*-The top node in a tree.
- *Child*-A node directly connected to another node when moving away from the Root.
- *Parent*-The converse notion of a *child*.
- *Sibling*-A group of nodes with the same parent.
- *Descendant*-A node reachable by repeated proceeding from parent to child.
- *Ancestor*-A node reachable by repeated proceeding from child to parent.
- *Leaf-(less commonly called External node)* A node with no children.
- *Branch*
- *Internal node*- A node with at least one child.

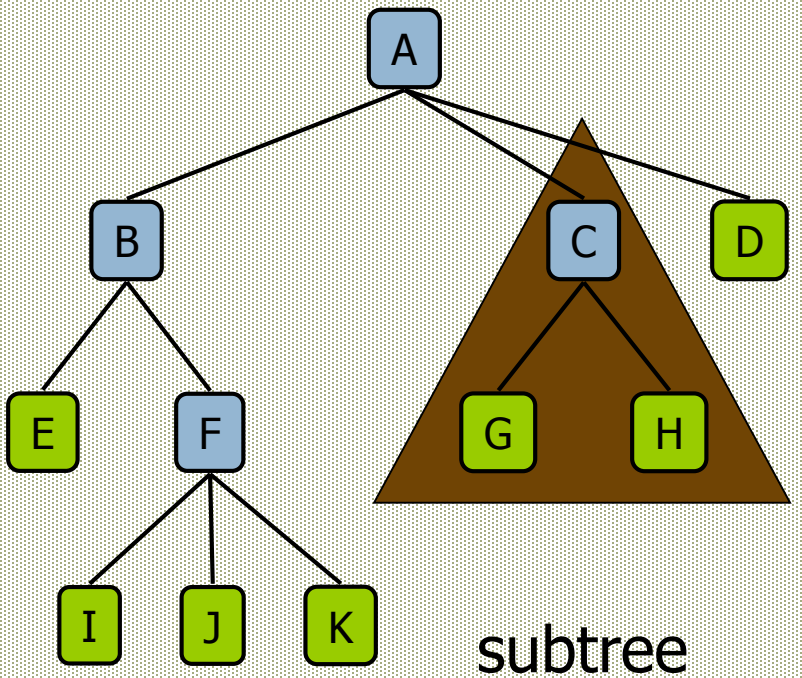
Tree Terminology

- *Degree*-The number of sub trees of a node.
- *Edge*-The connection between one node and another.
- *Path*-A sequence of nodes and edges connecting a node with a descendant.
- *Level*-The level of a node is defined by 1 + (the number of connections between the node and the root).
- *Height of node*-The height of a node is the number of edges on the longest path between that node and a leaf.
- *Height of tree*-The height of a tree is the height of its root node.
- *Depth*-The depth of a node is the number of edges from the tree's root node to the node.
- *Forest*-A forest is a set of $n \geq 0$ disjoint trees.

Tree Terminology

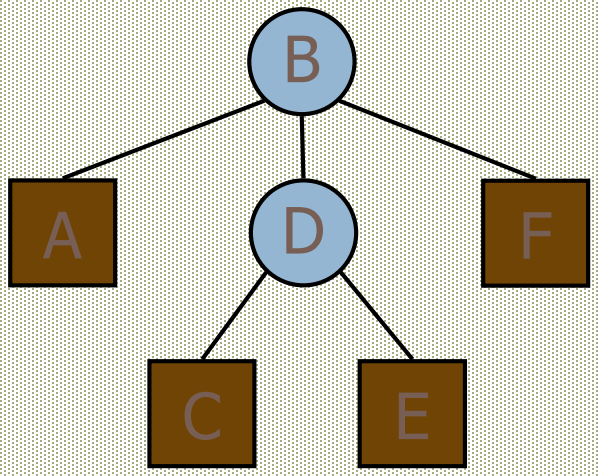
- **Root**: node without parent (A)
- **Siblings**: nodes share the same parent
  (B,C,D) are sibling sharing A
- **Internal node**: node with at least one child (A, B, C, F)
- **External node** (leaf): node without children (E, I, J, K, G, H, D)
- **Ancestors** of a node: parent, grandparent, grand-grandparent, etc.
- **Descendant** of a node: child, grandchild, grand-grandchild, etc.
- **Depth** of a node: number of ancestors
- **Height** of a tree: maximum depth of any node (3)
- **Degree** of a node: the number of its children
- **Degree** of a tree: the maximum number of its node.

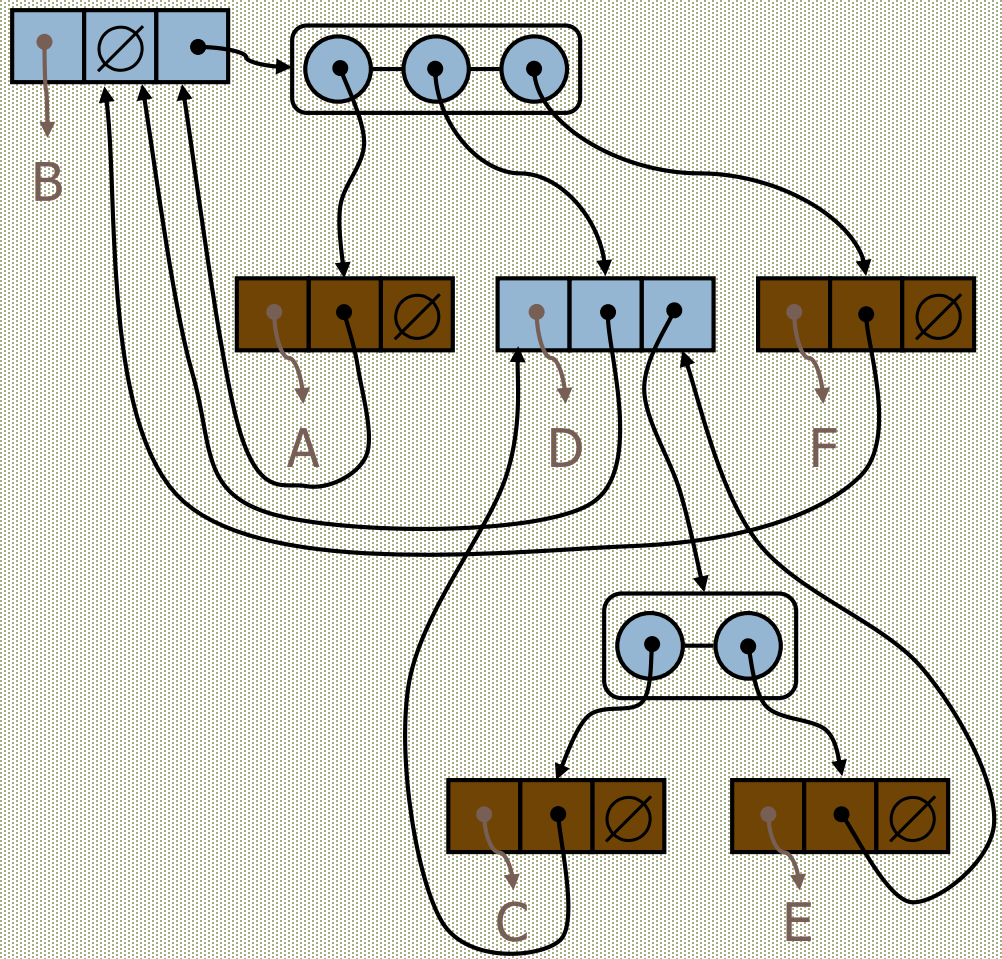
- **Subtree:** tree consisting of a node and its descendants



A Tree Representation

- A node is represented by an object storing
  - Element
  - Parent node
  - Sequence of children nodes





A Tree Representation

Representation Of Trees

- List Representation
  - we can write of Figure 5.2 as a list in which each of the subtrees is also a list

    (A(B(E(K,L),F),C(G),D(H(M),I,J)))
  - The root comes first, followed by a list of sub-trees

Figure 5.3: Possible list representation for trees

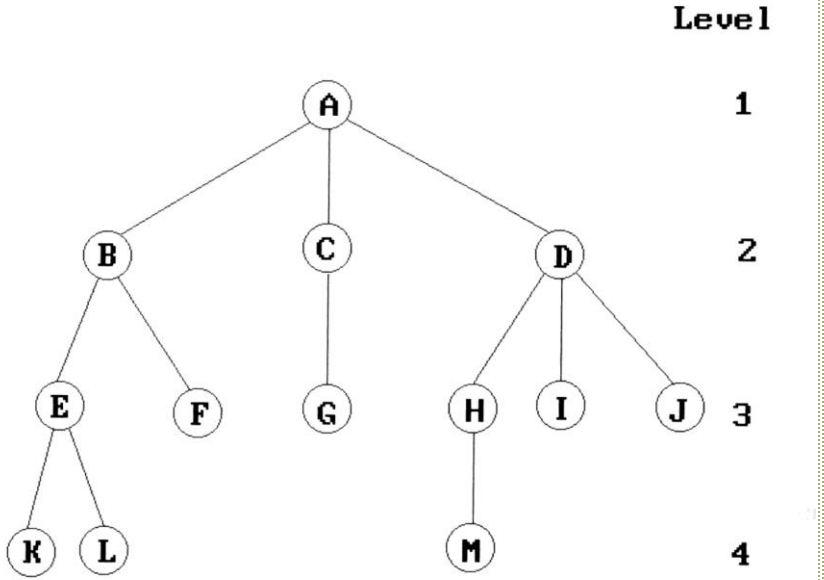


Figure 5.2: A sample tree

A Tree Representation

- Representation Of Trees (cont'd)
  - Left Child-Right Sibling Representation

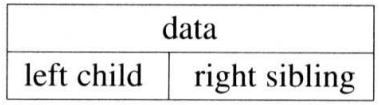


Figure 5.4: Left child-right sibling node structure

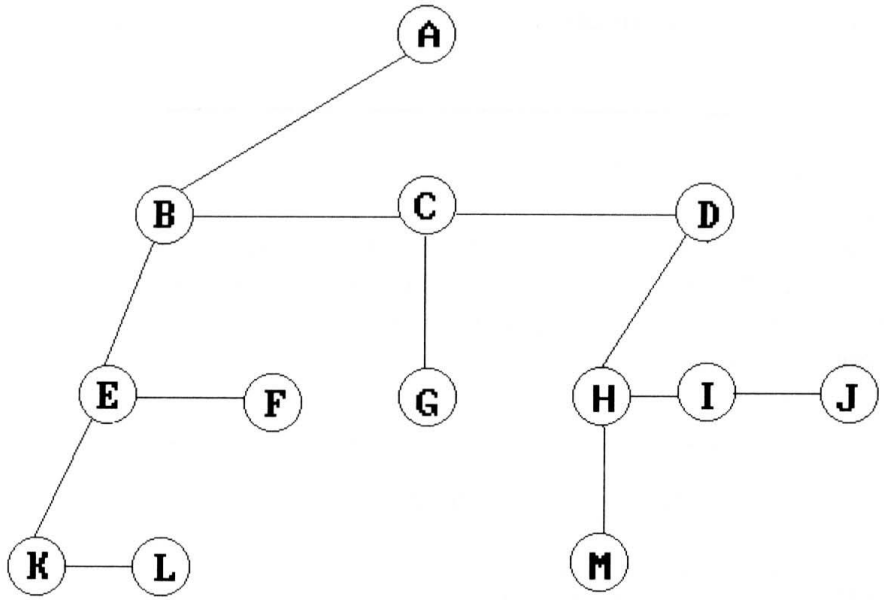


Figure 5.5: Left child-right sibling representation of a tree

Tree Traversal Algorithms

- A ***traversal*** of a tree *T* is a systematic way of accessing, or "visiting," all the positions of *T*.
- The specific action associated with the "visit" of a position *p* depends on the application of this traversal.
- Two main schemes involved in general trees:
  - Preorder Traversals
  - Postorder Traversals

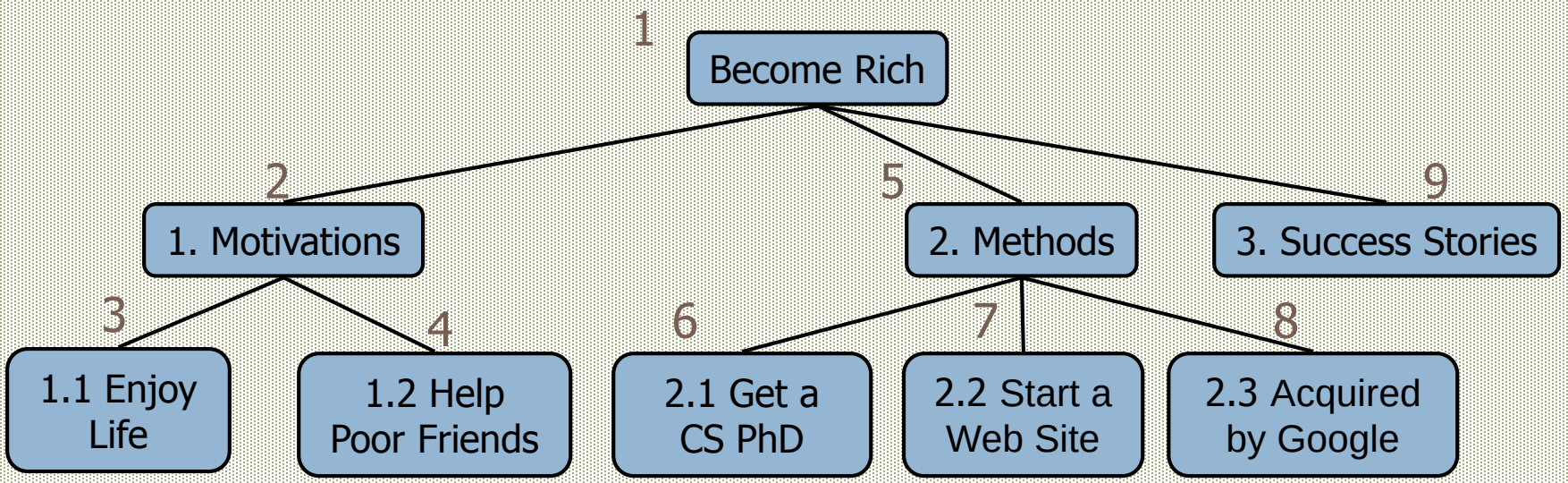
Tree Traversal Algorithms

- Two main methods:
  - Preorder
  - Postorder
- Preorder:
  - visit the root
  - traverse in preorder the children (subtrees)
- Postorder
  - traverse in postorder the children (subtrees)
  - visit the root

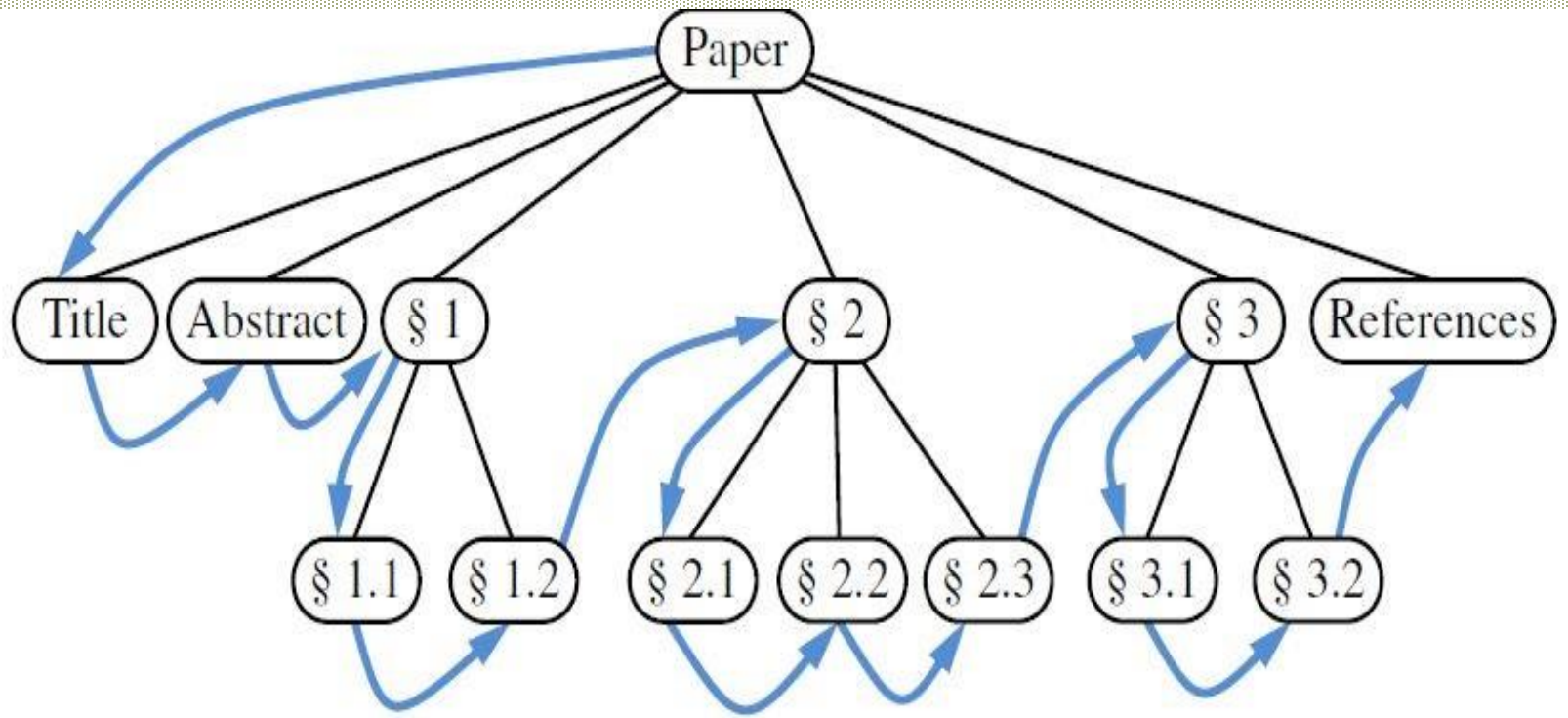
Preorder Traversal 1

- A traversal visits the nodes of a tree in a systematic manner
- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

Algorithm *preOrder*(*v*)
  visit(*v*)
  for each child *w* of *v*
    preorder (*w*)



Preorder Traversals 2

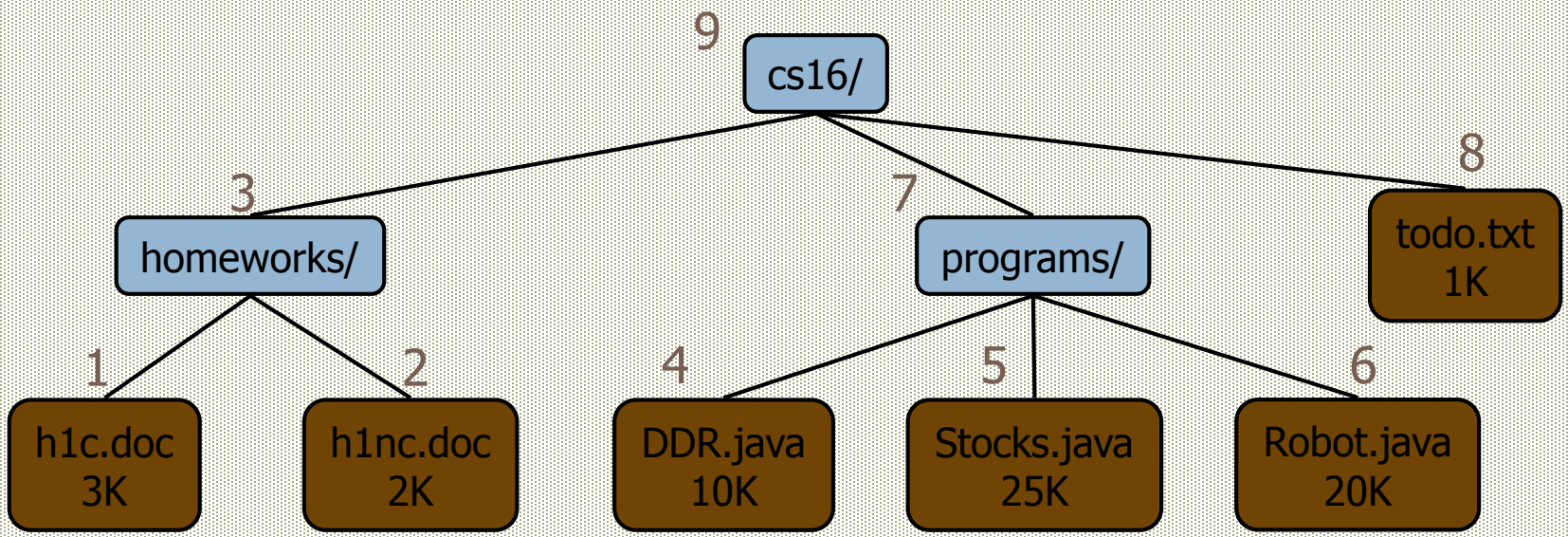


Preorder traversal of an ordered tree, where the children of each position are ordered from left to right.

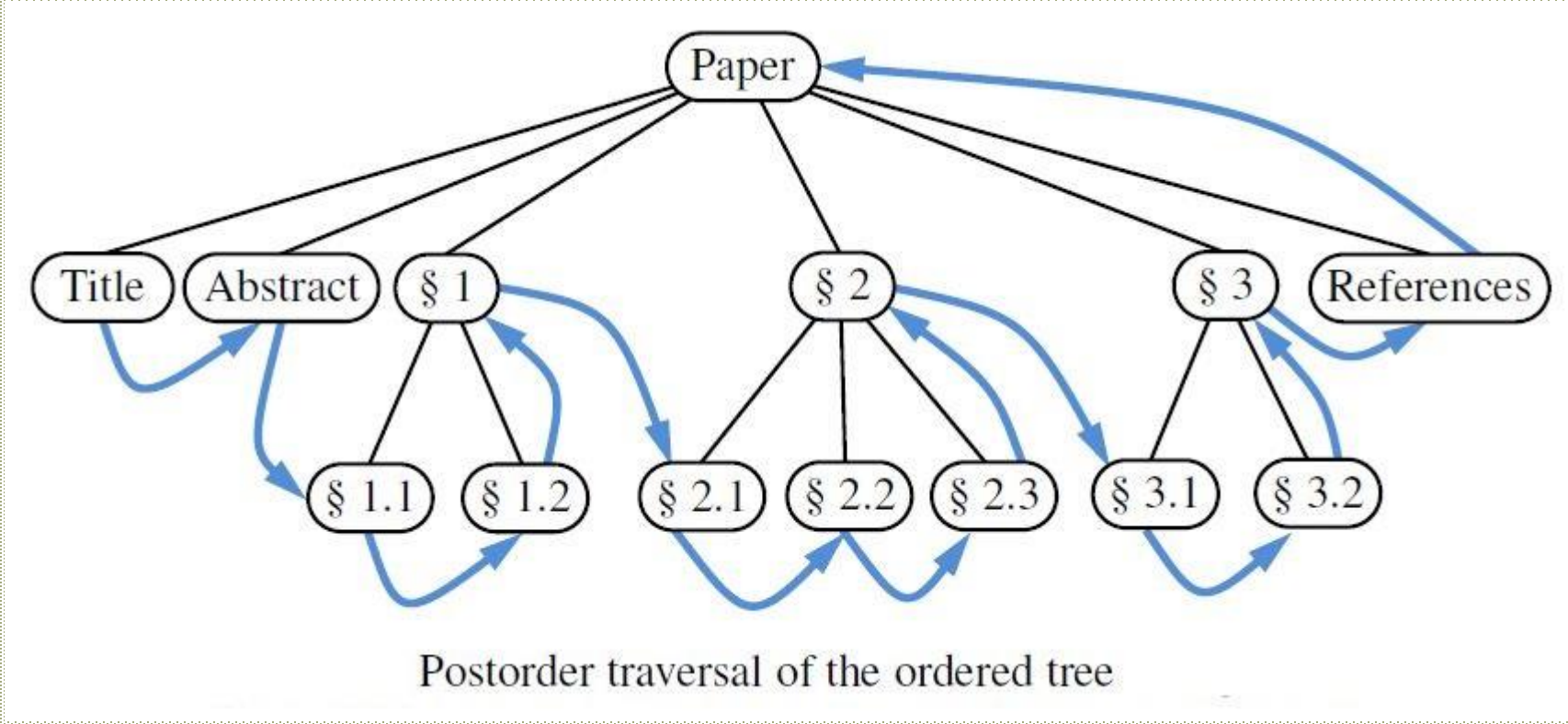
Postorder Traversal 1

- In a postorder traversal, a node is visited after its descendants
- Application: compute space used by files in a directory and its subdirectories

Algorithm *postOrder*(*v*)
  for each child *w* of *v*
    postOrder (*w*)
  visit(*v*)



Postorder Traversals 2

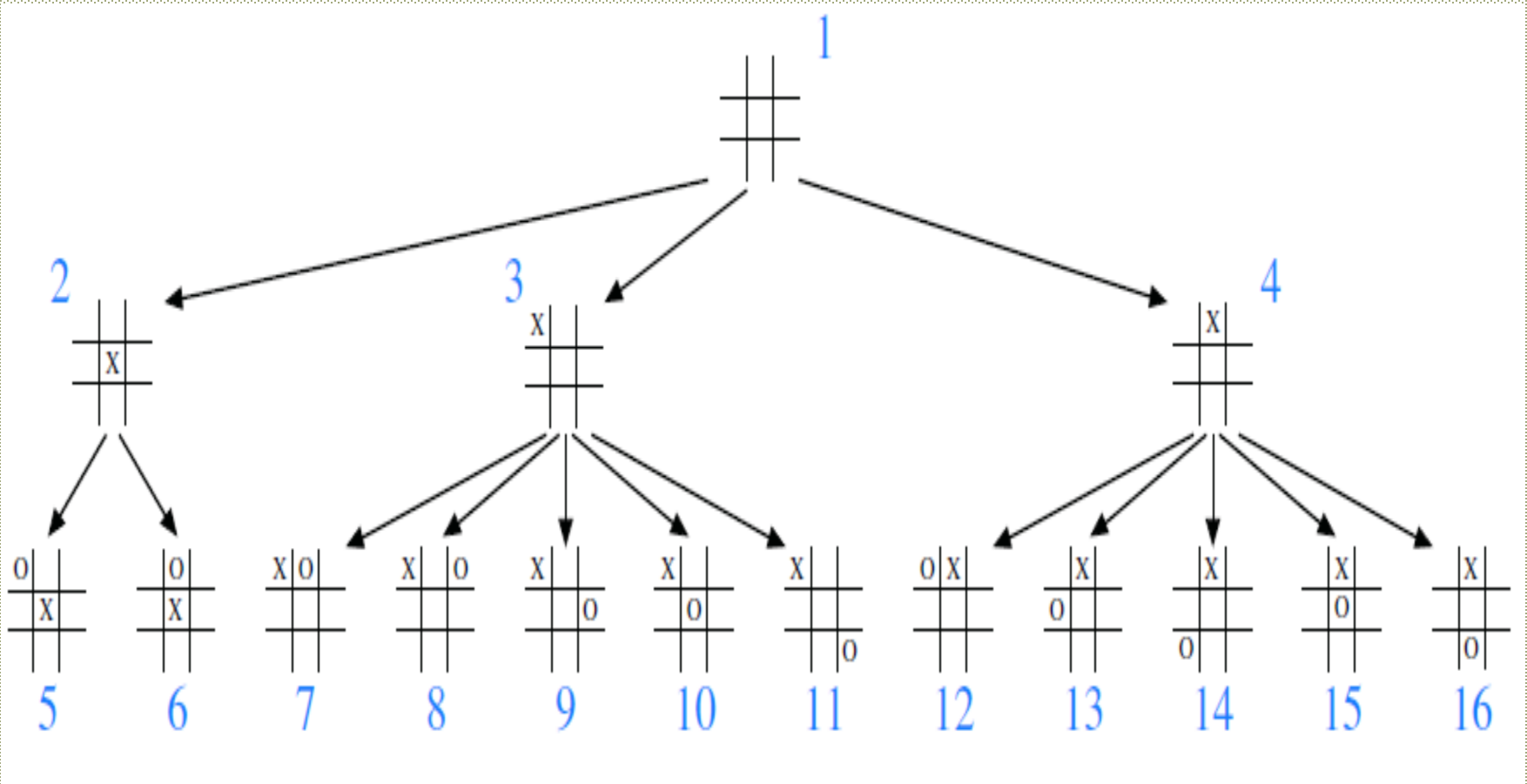


Postorder traversal of the ordered tree

Breadth-First Tree Traversal

- Another approach is to traverse a tree so that we visit all the positions at depth *d* before we visit the positions at depth *d*+1.
- Such an algorithm is known as a ***breadth-first traversal***.
- However preorder and postorder traversals are common ways of visiting the positions of a tree,

Breadth-First Tree Traversal



Breadth-First Tree Traversal

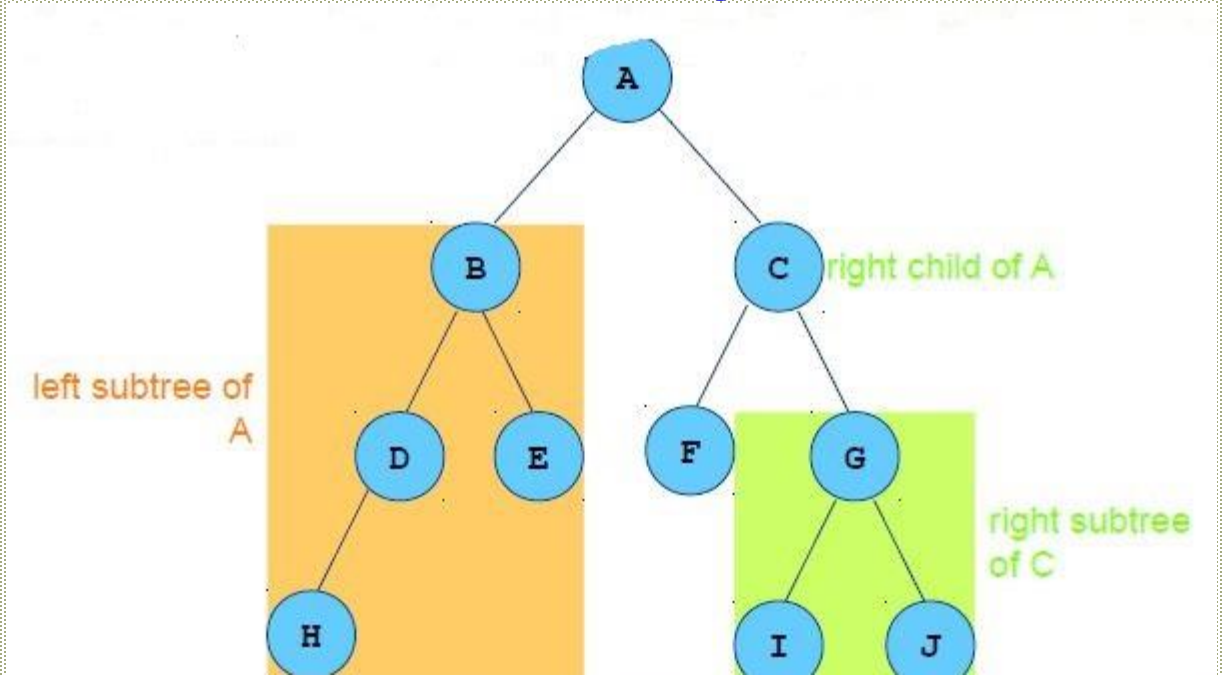
- We use a queue to produce a FIFO (i.e., first-in first-out) semantics for the order in which we visit nodes.
- The overall running time is $O(n)$, due to the n calls to enqueue and n calls to dequeue.

Running-Time Analysis

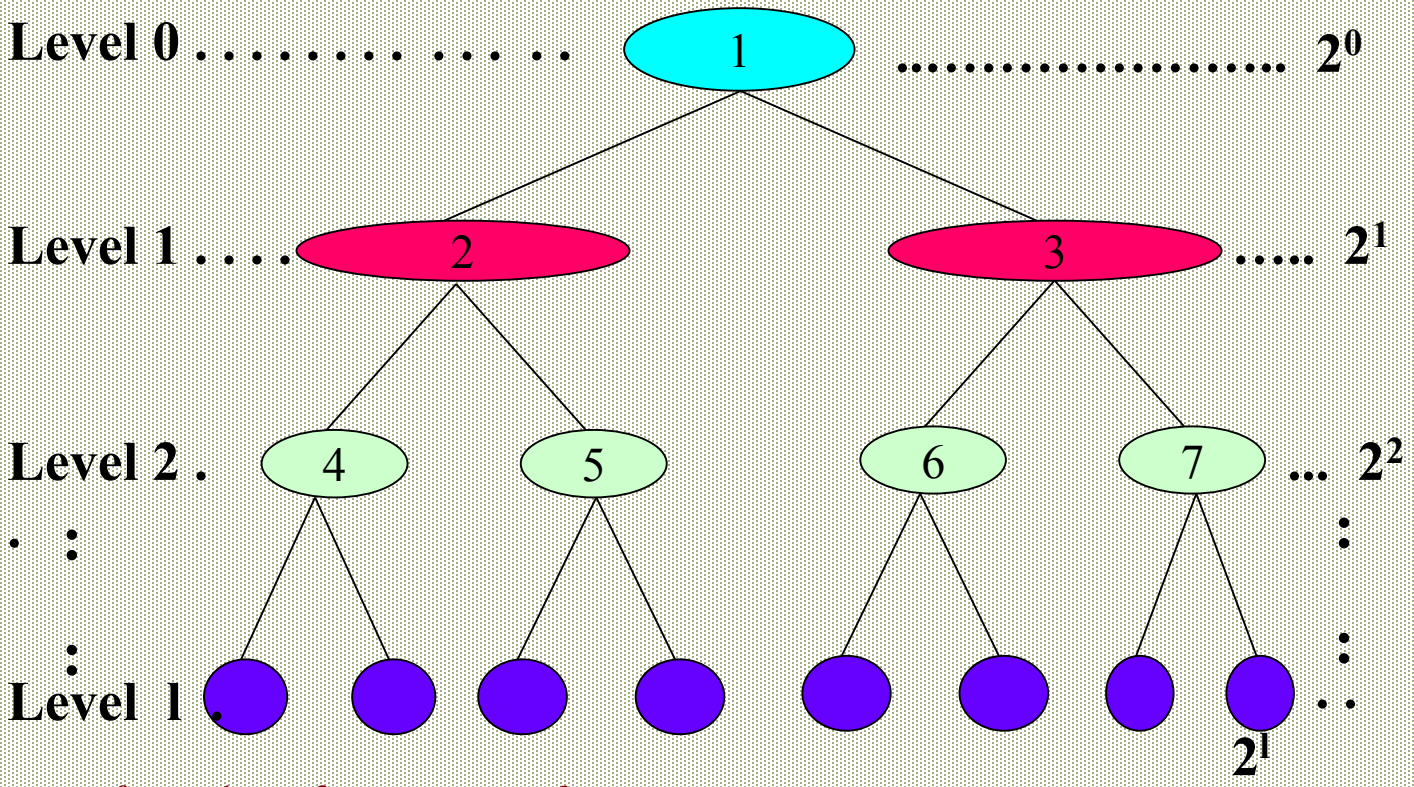
- Both preorder and postorder traversal algorithms are efficient ways to access all the positions of a tree.
- At each position p, the nonrecursive part of the traversal algorithm requires time $O(c_p+1)$, where c_p is the number of children of p, under the assumption that the "visit" itself takes $O(1)$ time.
- The overall running time for the traversal of tree T is $O(n)$, where n is the number of positions in the tree.
- This running time is asymptotically optimal since the traversal must visit all n positions of the tree.

Binary Trees

- A finite (possibly empty) collection of elements
- A nonempty binary tree has a root element and the remaining elements (if any) are partitioned into two binary trees
- They are called the left and right subtrees of the binary tree



Depth (Height) Calculation in a full binary tree of N vertices



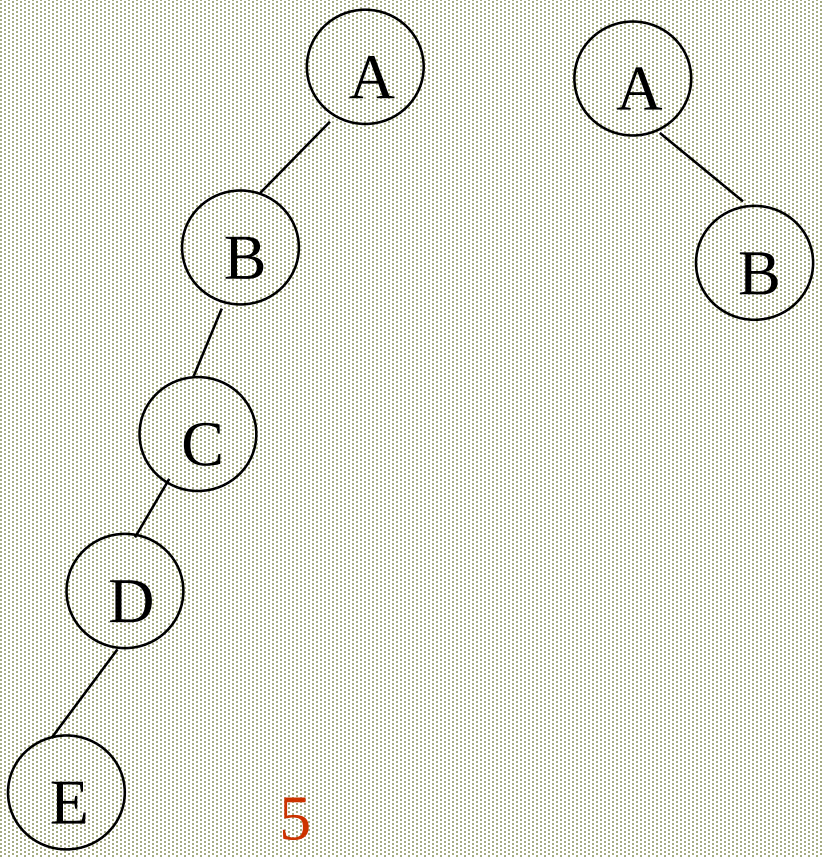
$2^0 + 2^1 + 2^2 + \ldots + 2^l = N$ in case of a full binary tree

BinaryTree ADT

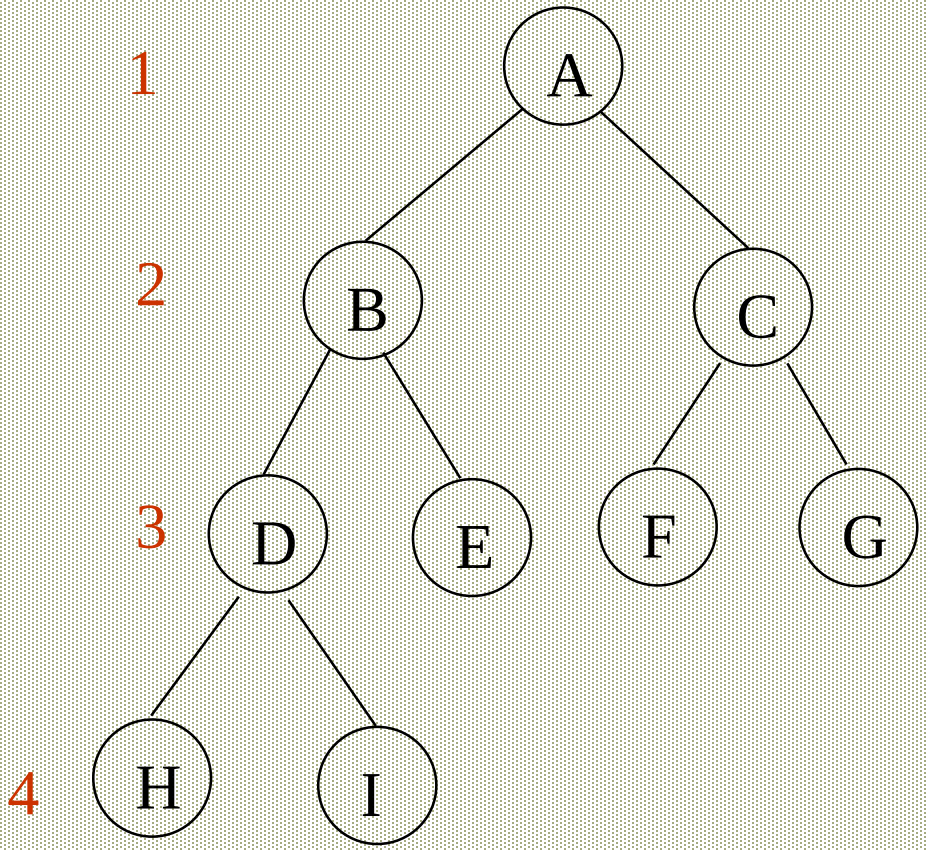
- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Additional methods:
  - position leftChild(p)
  - position rightChild(p)
  - position sibling(p)
- Update methods may be defined by data structures implementing the BinaryTree ADT

Examples of the Binary Tree

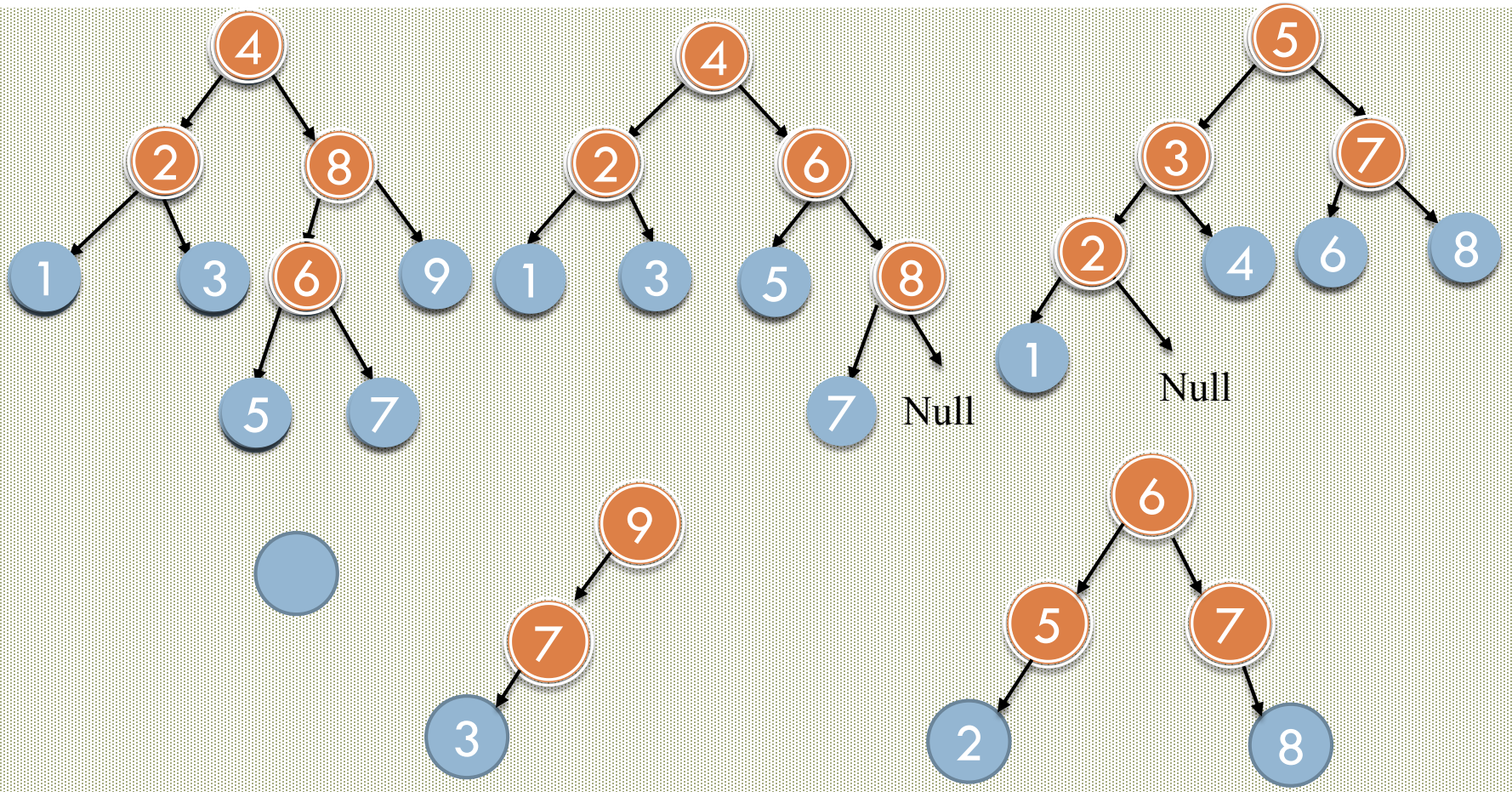
Skewed Binary Tree



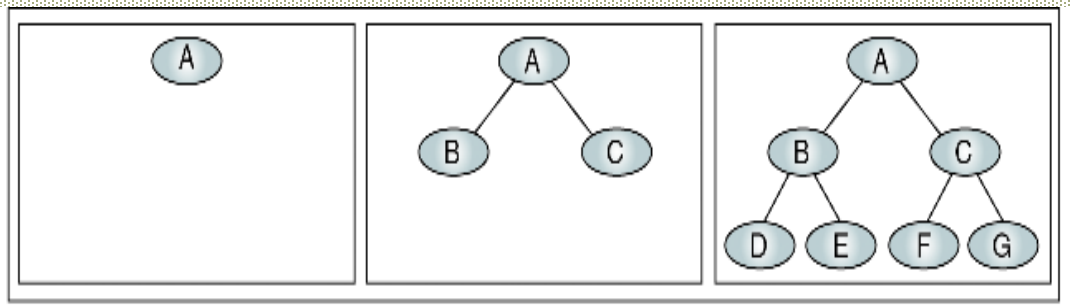
Complete Binary Tree



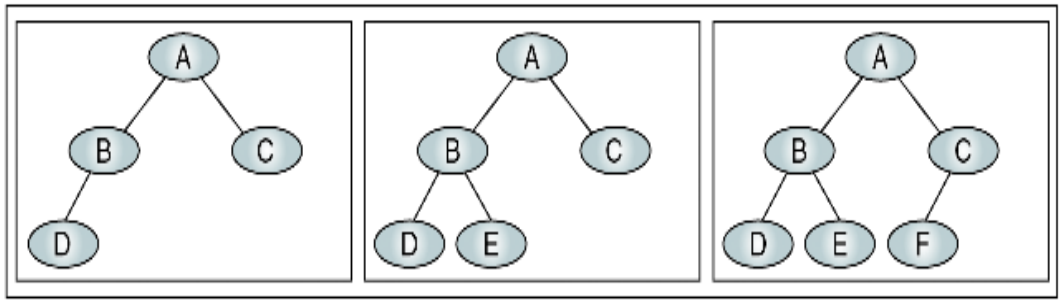
Binary Trees Examples



Other Examples of Binary Trees



(a) Complete trees (at levels 0, 1, and 2)

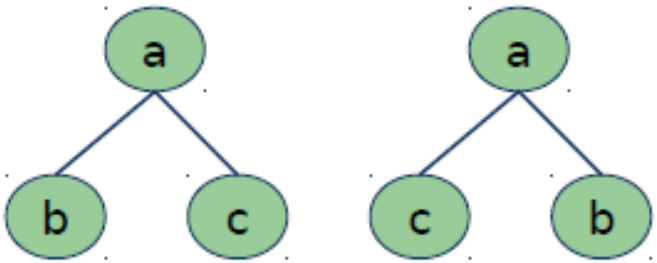


(b) Nearly complete trees (at level 2)

FIGURE 6-7 Complete and Nearly Complete Trees

Difference Between a Tree & a Binary Tree

- A binary tree may be empty; a tree cannot be empty.
- No node in a binary tree may have a degree more than 2, whereas there is no limit on the degree of a node in a tree.
- The subtrees of a binary tree are ordered; those of a tree are not ordered.



- different when viewed as a binary tree

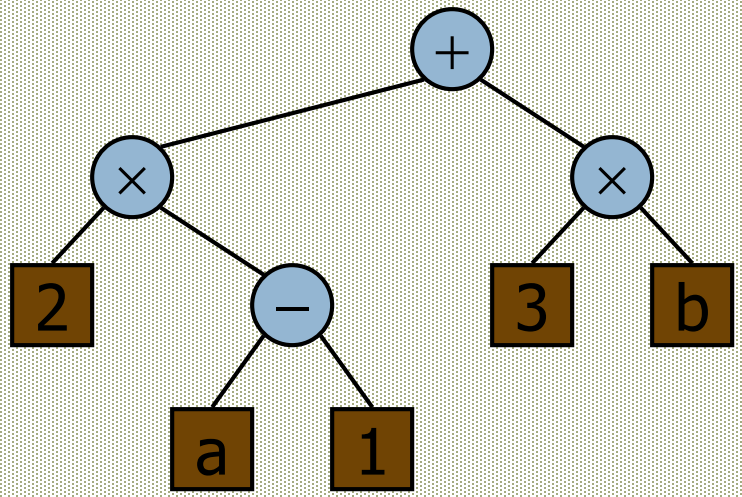
- same when viewed as a tree

Applications:

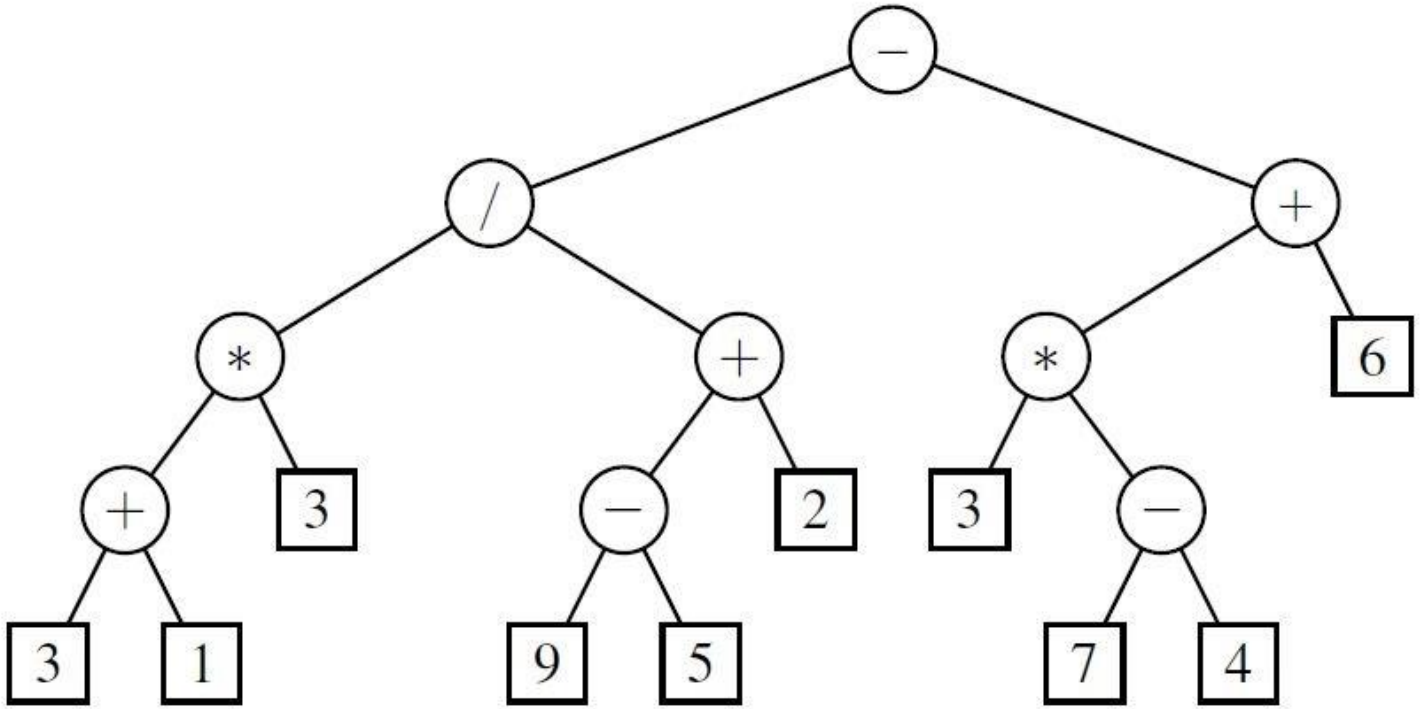
- arithmetic expressions
- decision processes
- searching

Arithmetic Expression Tree

- Binary tree are associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands
- Example: arithmetic expression tree for the expression $(2 \times (a - 1) + (3 \times b))$



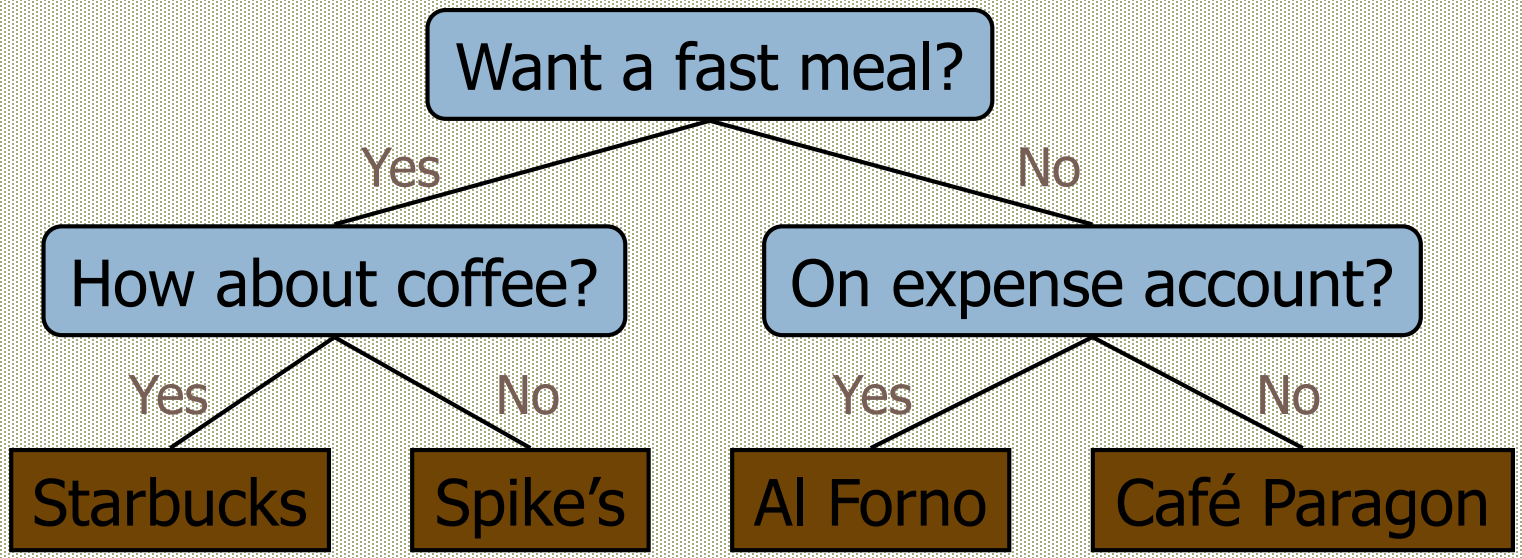
Arithmetic Expression Tree



A binary tree representing an arithmetic expression. This tree represents the expression $((((3+1)*3)/((9-5)+2))-((3*(7-4))+6))$. The value associated with the internal node labeled "/" is 2.

Decision Tree

- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision



Maximum Number of Nodes in a Binary Tree

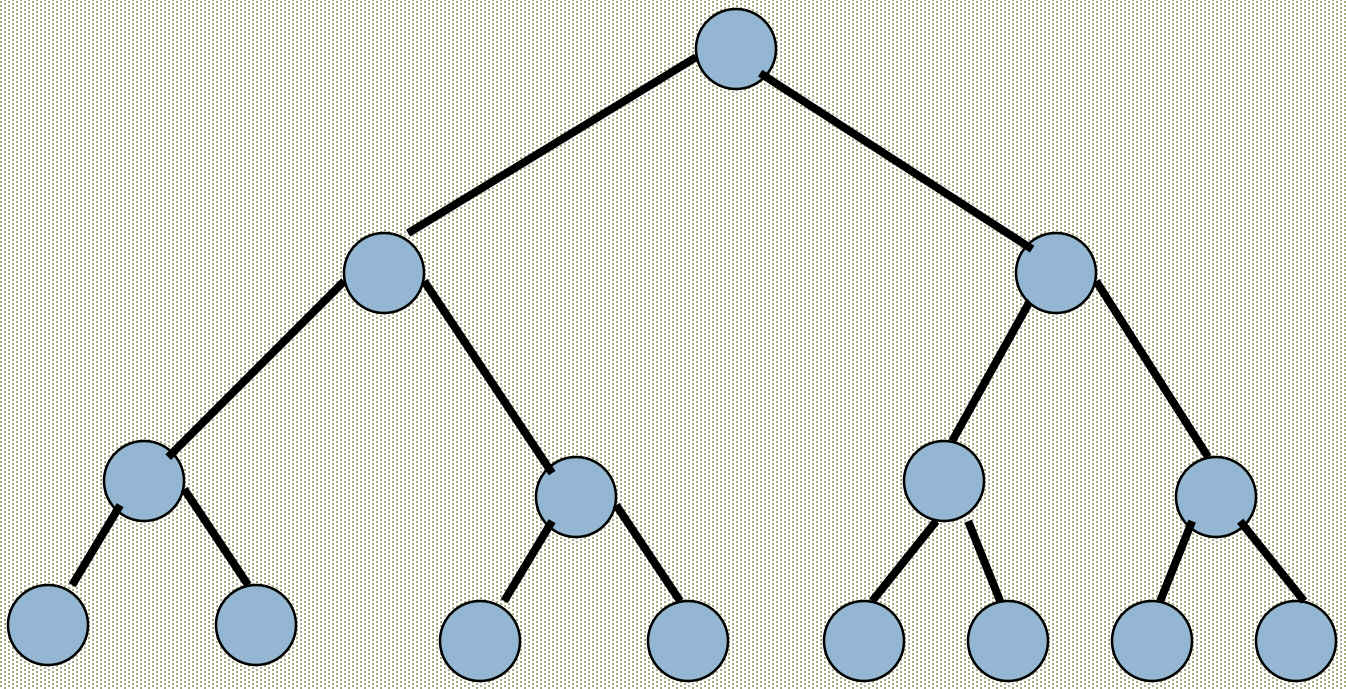
- The maximum number of nodes on depth i of a binary tree is 2^i, i>=0.
- The maximum number of nodes in a binary tree of height k is $2^{k+1}-1$, k>=0.

Prove by induction.

$$\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$$

Full Binary Tree

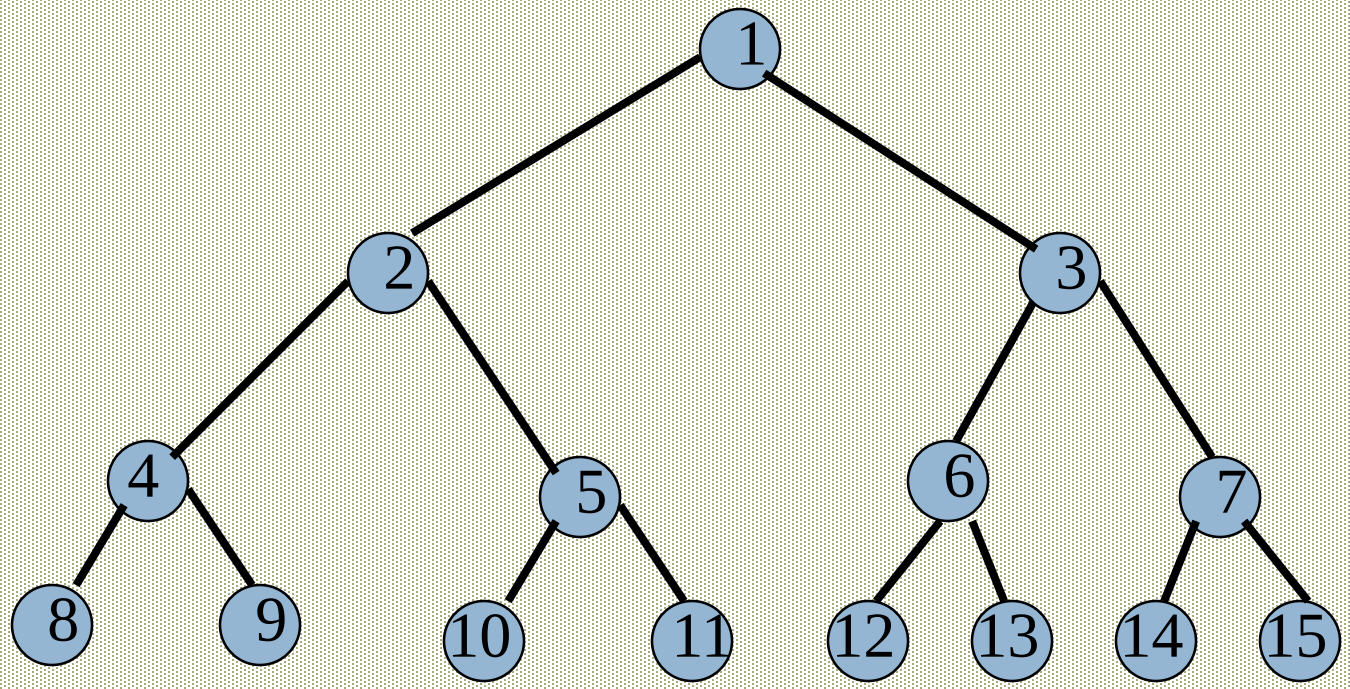
- A full binary tree of a given height k has $2^{k+1}-1$ nodes.



Height 3 full binary tree.

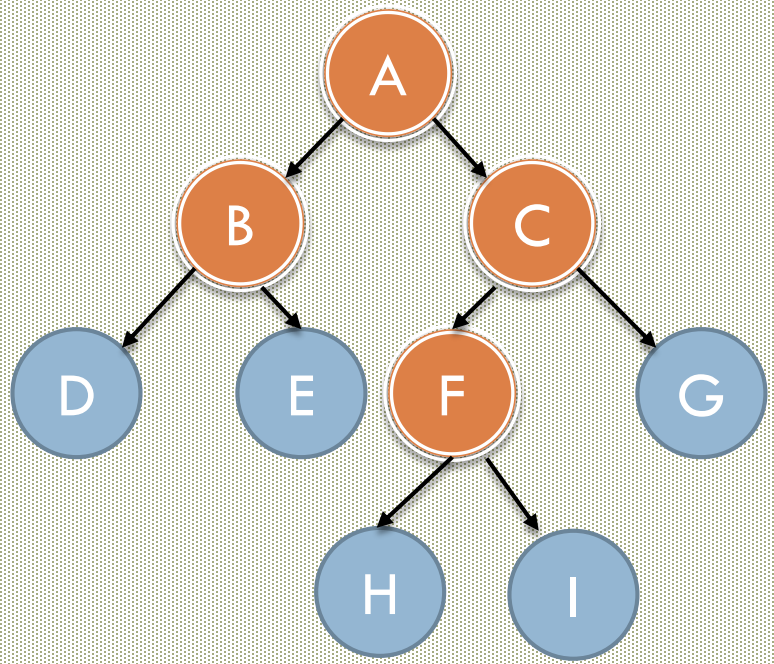
Labeling Nodes In A Full Binary Tree

- Label the nodes 1 through $2^{k+1} - 1$.
- Label by levels from top to bottom.
- Within a level, label from left to right.



Binary Tree (Strict/Proper binary Trees)

- A binary tree is a strict or proper binary tree when each node can have either 2 or 0 children



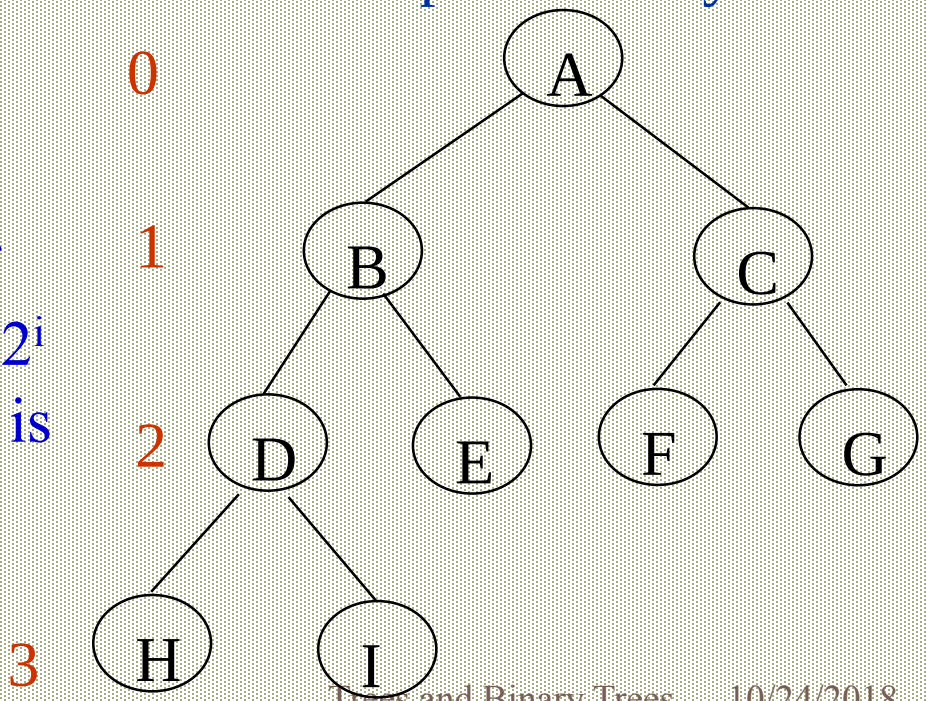
Binary Tree (Strict/Proper binary Trees)

- The subtree rooted at a left or right child of an internal node v is called a **left subtree** or **right subtree,** respectively, of v.
- It can also be referred to as **full** binary tree
- Thus, in a proper binary tree, every internal node has exactly two children.
- A binary tree that is not proper is **improper.**

Complete Binary Tree

- A binary tree is said to be complete if all levels except possibly the last is completely filled and all nodes are as far left as possible

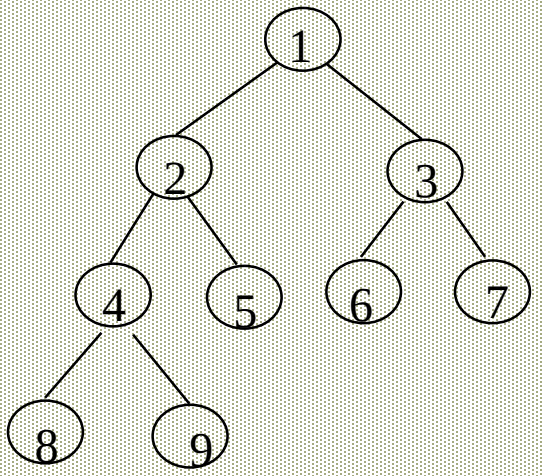
Complete Binary Tree



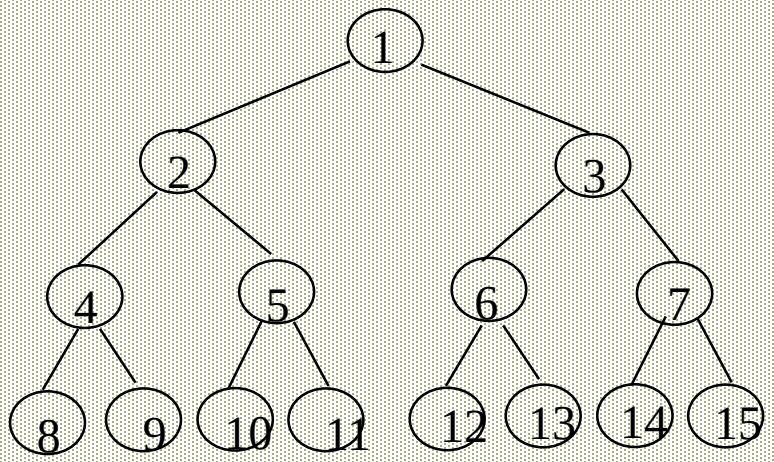
We can get the maximum number of nodes at any level i $=2^i$
Height of a complete is obtained using $\log_2 n$

Complete Binary Trees

- A labeled binary tree containing the labels 1 to n with root 1, branches leading to nodes labeled 2 and 3, branches from these leading to 4, 5 and 6, 7, respectively, and so on.
- A binary tree with *n* nodes and level *k* is complete *iff* its nodes correspond to the nodes numbered from 1 to *n* in the full binary tree of level *k*.



Complete binary tree



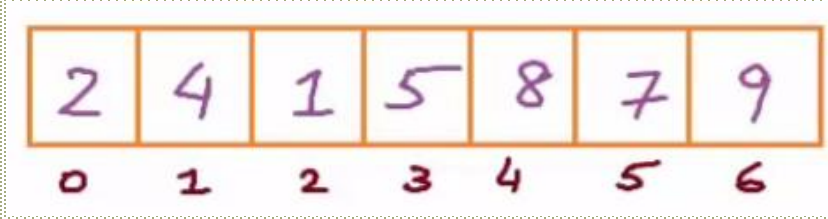
Full binary tree of depth 3

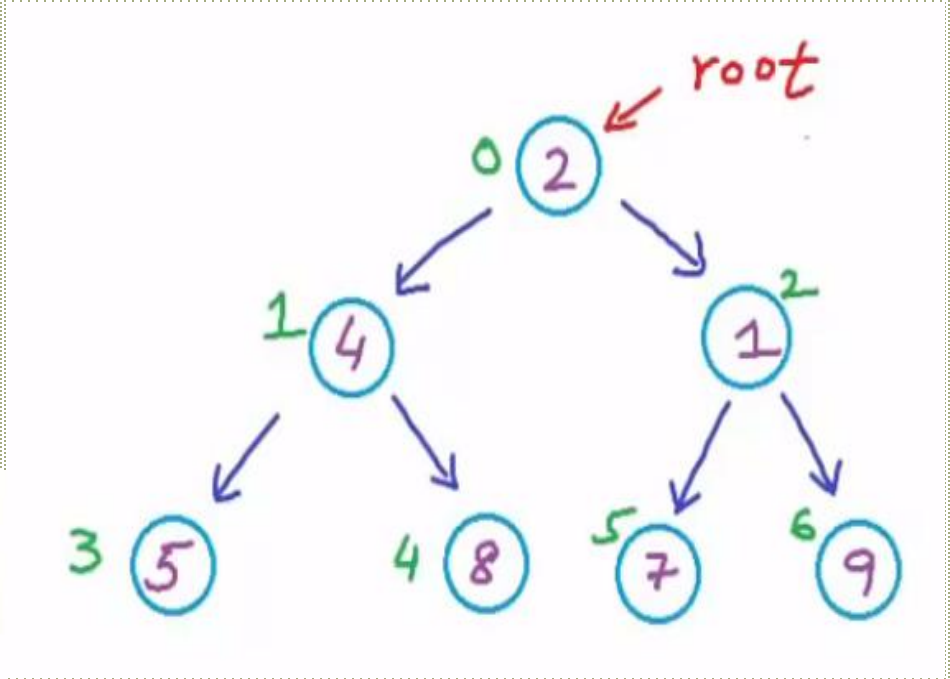
Binary Tree Representation

- Array representation
- Linked representation

Array Representation of Binary Tree

- The binary tree is represented in an array by storing each element at the array position corresponding to the number assigned to it.

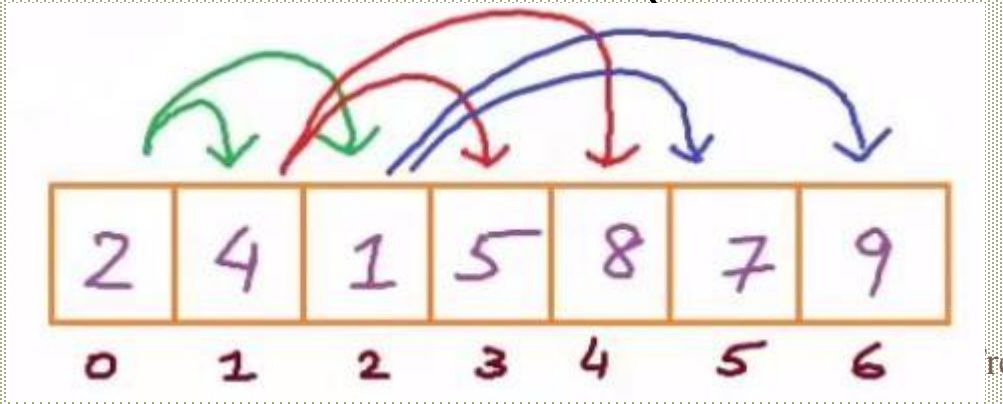




Array Representation of Binary Tree

- Array index from 0
- Ex. To find node's 2 left and right child
- Left Child's Index = 2 * Parent's Index + 1
- Right Child's Index = 2 * Parent's Index + 2
- To find a node's parent use this formula:

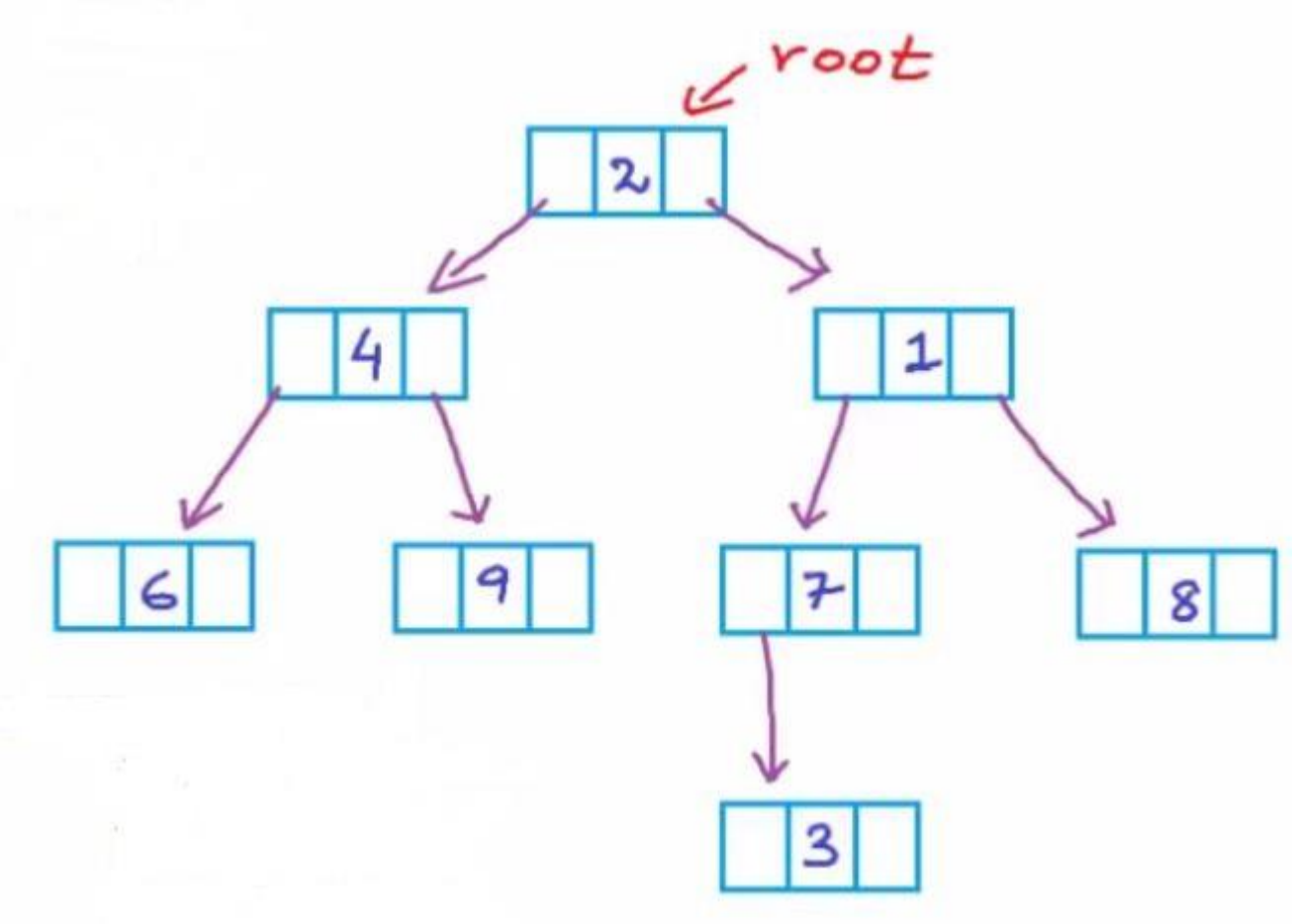
  Parent's Index = (Child's Index – 1) / 2



Linked Representation of Binary Tree

- The most popular way to present a binary tree
- Each element is represented by a node that has two link fields (leftChild and rightChild) plus an element field
- Each binary tree node is represented as an object whose data type is binaryTreeNode
- The space required by an *n* node binary tree is *n* * sizeof(binaryTreeNode)

Linked Representation of Binary Tree



Common Binary Tree Operations

- Determine the height
- Determine the number of nodes
- Make a copy
- Determine if two binary trees are identical
- Display the binary tree
- Delete a tree
- If it is an expression tree, evaluate the expression
- If it is an expression tree, obtain the parenthesized form of the expression

Binary Tree Traversal

- A binary tree traversal requires that each node of the tree be processed once and only once in a predetermined sequence.
- In the depth-first traversal processing process along a path from the root through one child to the most distant descendant of that first child before processing a second child.

Binary Tree Traversal

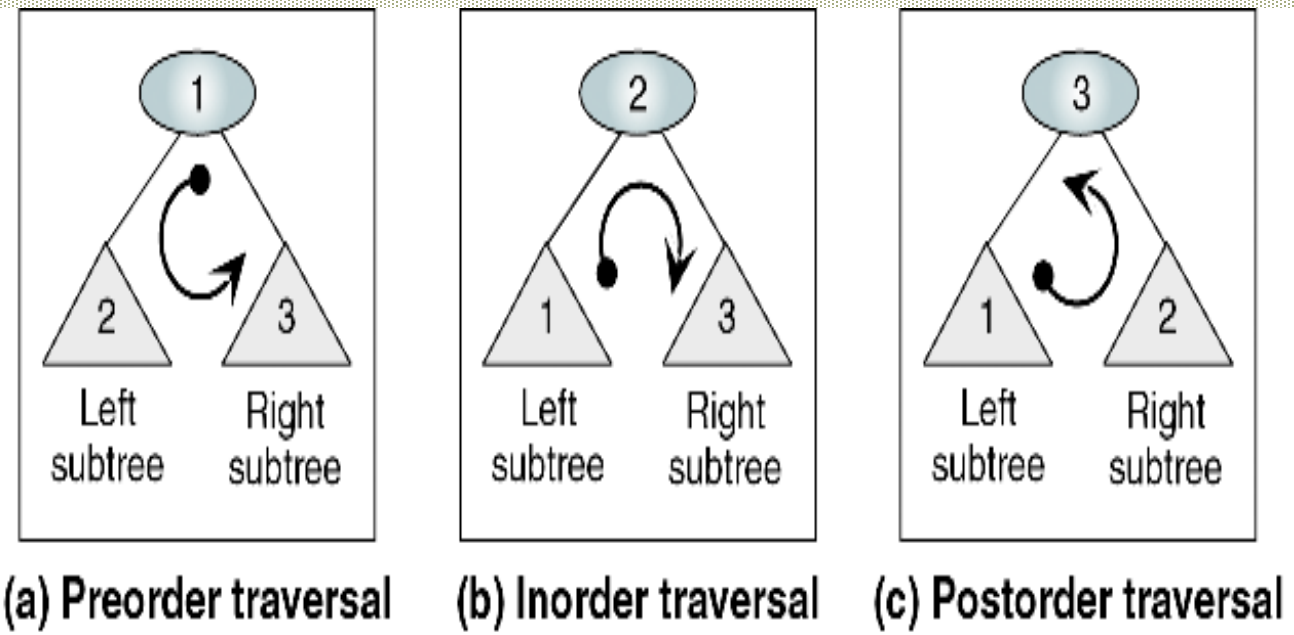


FIGURE 6-8 Binary Tree Traversals

Preorder Traversal of a Binary Tree

ALGORITHM 6-2 Preorder Traversal of a Binary Tree

```
Algorithm preOrder (root)
Traverse a binary tree in node-left-right sequence.
   Pre  root is the entry node of a tree or subtree
   Post each node has been processed in order
1  if (root is not null)
   1   process (root)
   2   preOrder (leftSubtree)
   3   preOrder (rightSubtree)
2  end if
end preOrder
```

Preorder Traversal of a Binary Tree

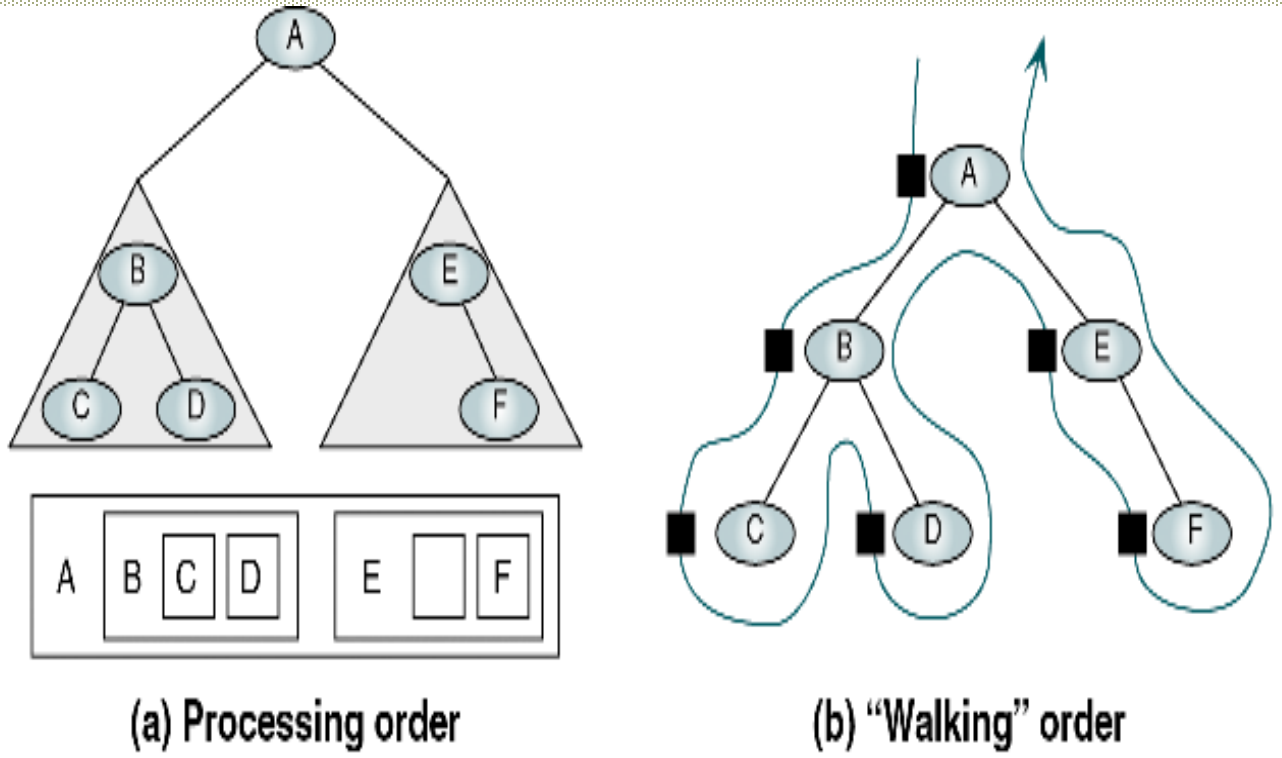


FIGURE 6-10 Preorder Traversal—A B C D E F

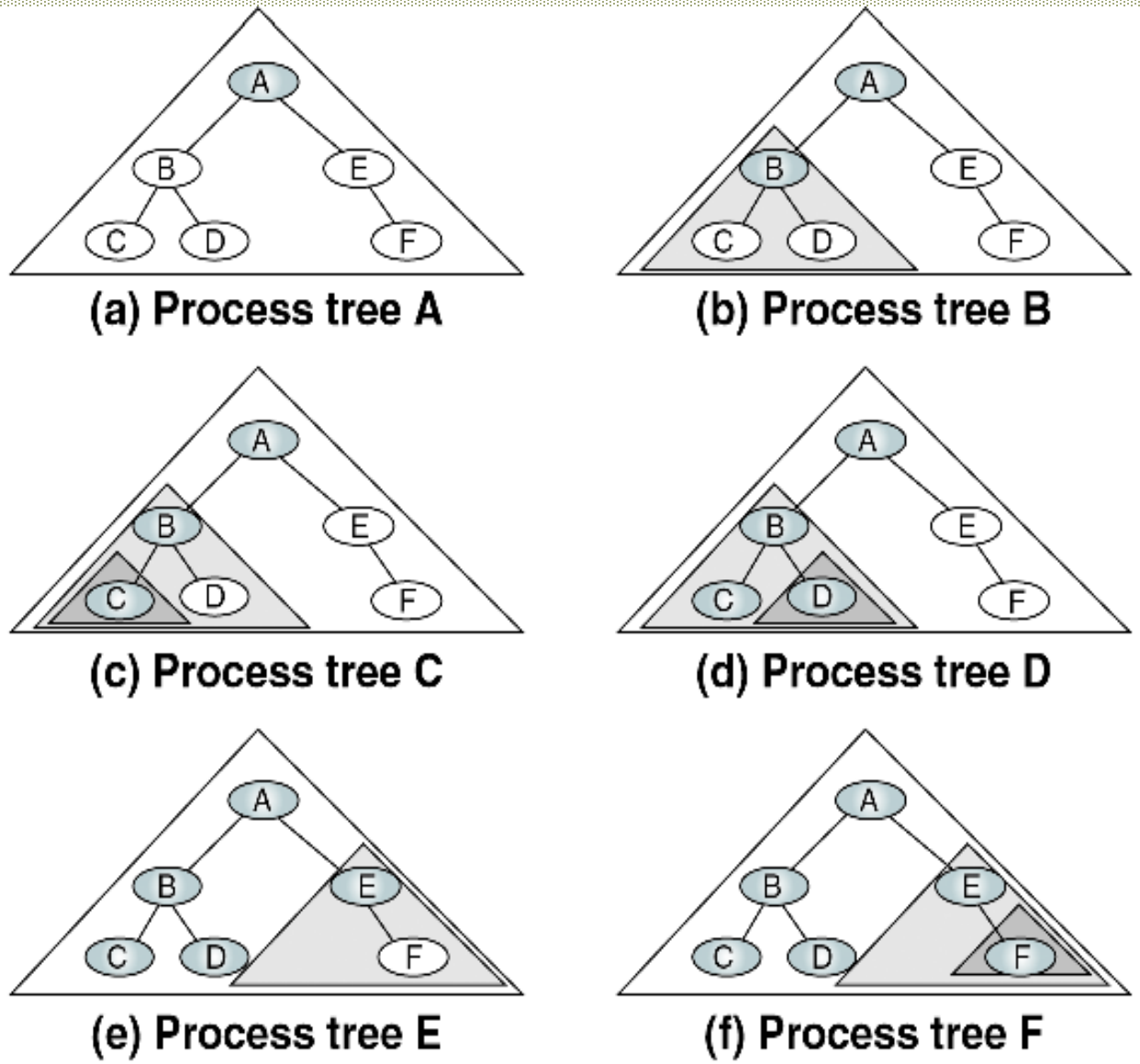
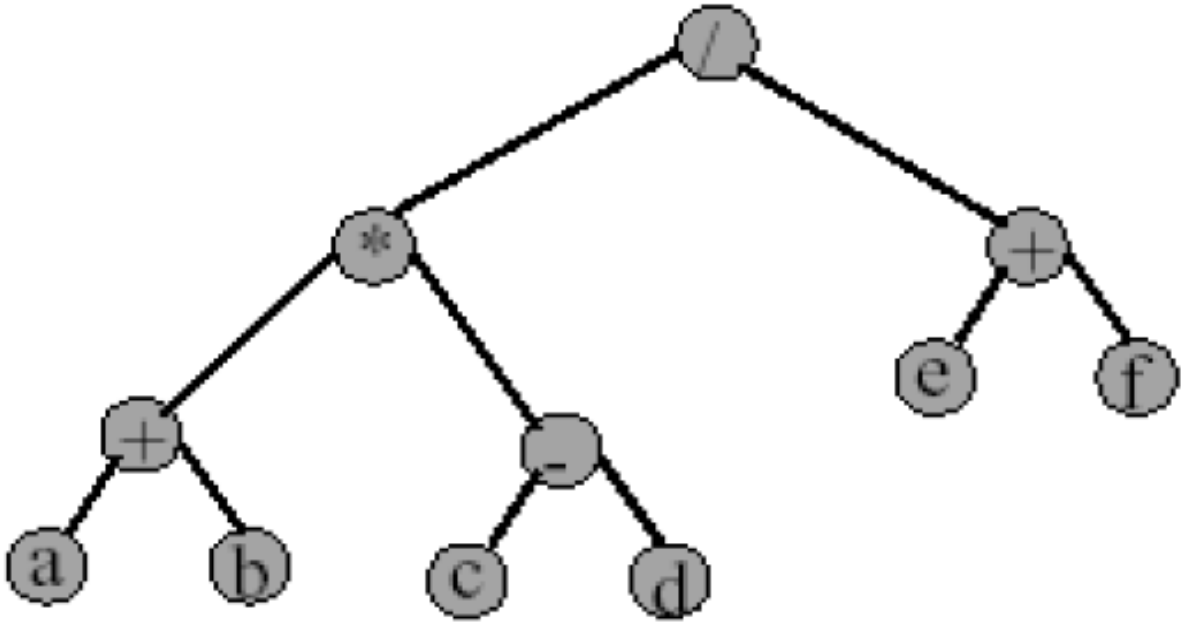


FIGURE 6-11 Algorithmic Traversal of Binary Tree

Preorder of Expression Tree



/ * + a b - c d + e f

Gives prefix form of expression.

Inorder Traversal of a Binary Tree

ALGORITHM 6-3 Inorder Traversal of a Binary Tree

```
Algorithm inOrder (root)
Traverse a binary tree in left-node-right seque
   Pre  root is the entry node of a tree or subt
   Post each node has been processed in order
1  if (root is not null)
   1   inOrder (leftSubTree)
   2   process (root)
   3   inOrder (rightSubTree)
2  end if
end inOrder
```

Inorder Traversal of a Binary Tree

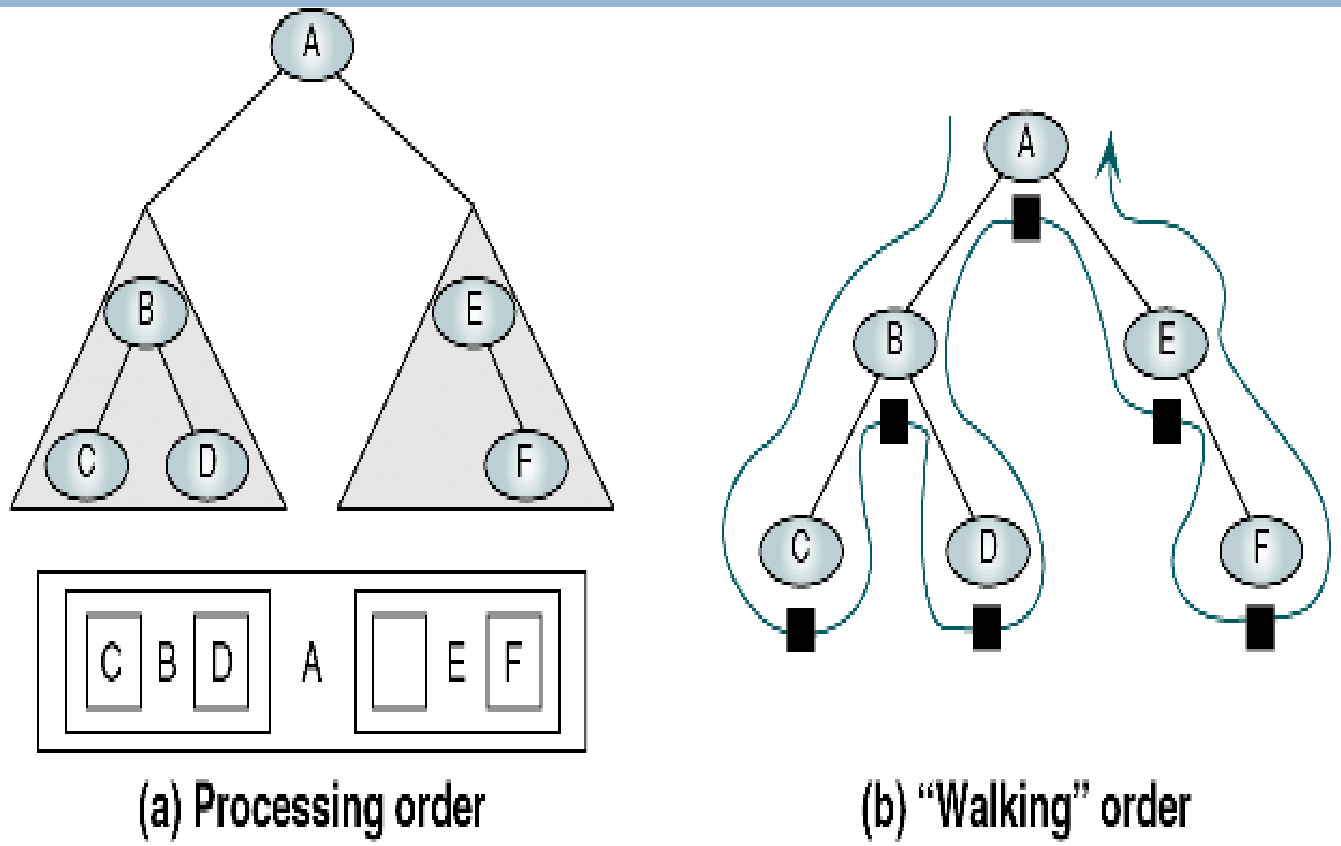
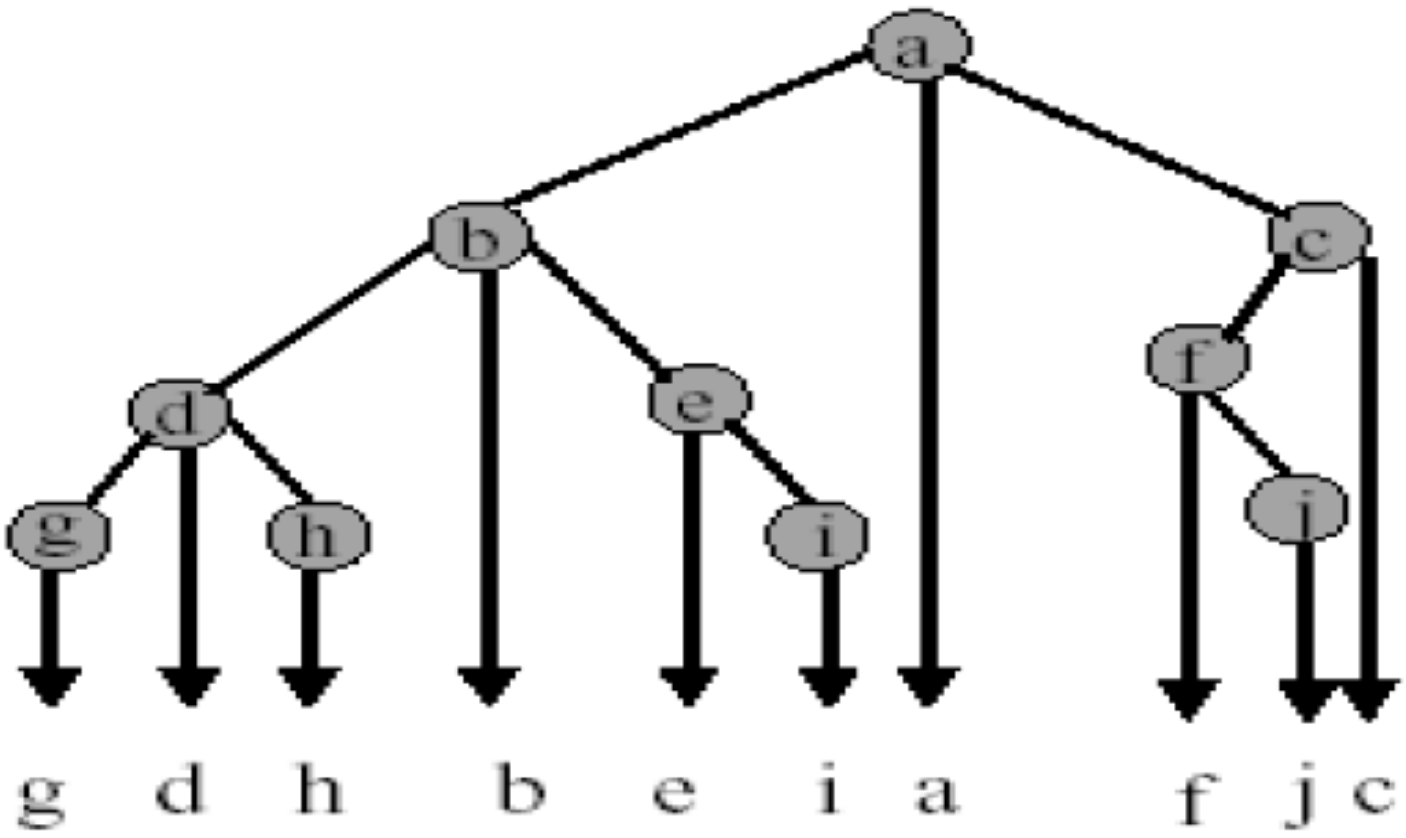
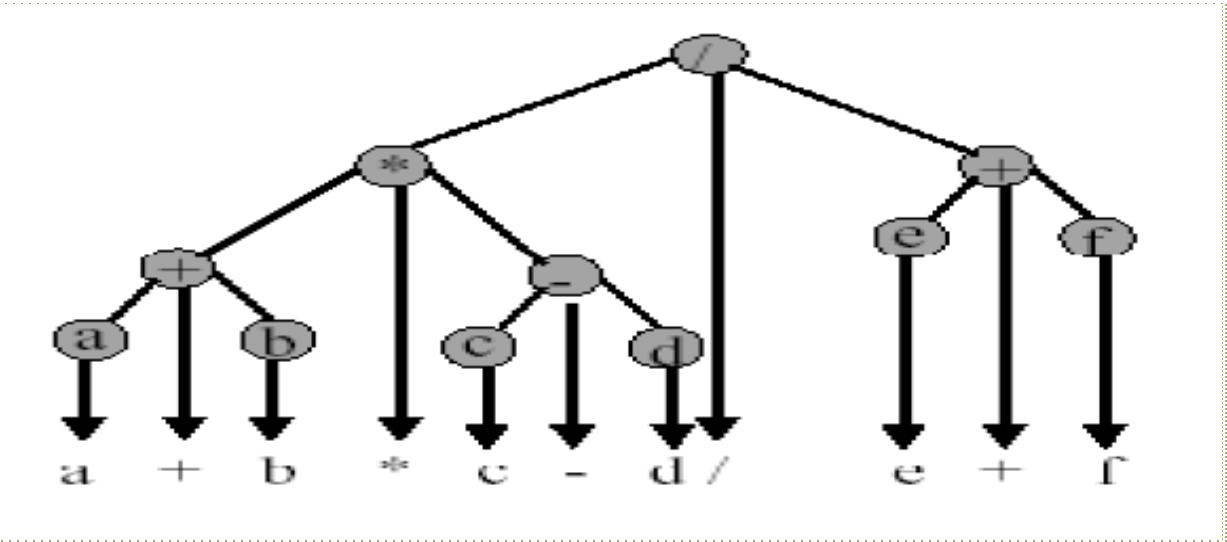


FIGURE 6-12 Inorder Traversal—C B D A E F

Inorder by Projection



Inorder of Expression Tree



- Gives infix form of expression, which is how we normally write math expressions. What about parentheses?

Postorder Traversal of a Binary Tree

ALGORITHM 6-4 Postorder Traversal of a Binary Tree

```
Algorithm postOrder (root)
Traverse a binary tree in left-right-node sequence.
   Pre  root is the entry node of a tree or subtree
   Post each node has been processed in order
1  if (root is not null)
   1   postOrder (left subtree)
   2   postOrder (right subtree)
   3   process (root)
2  end if
end postOrder
```

Postorder Traversal of a Binary Tree

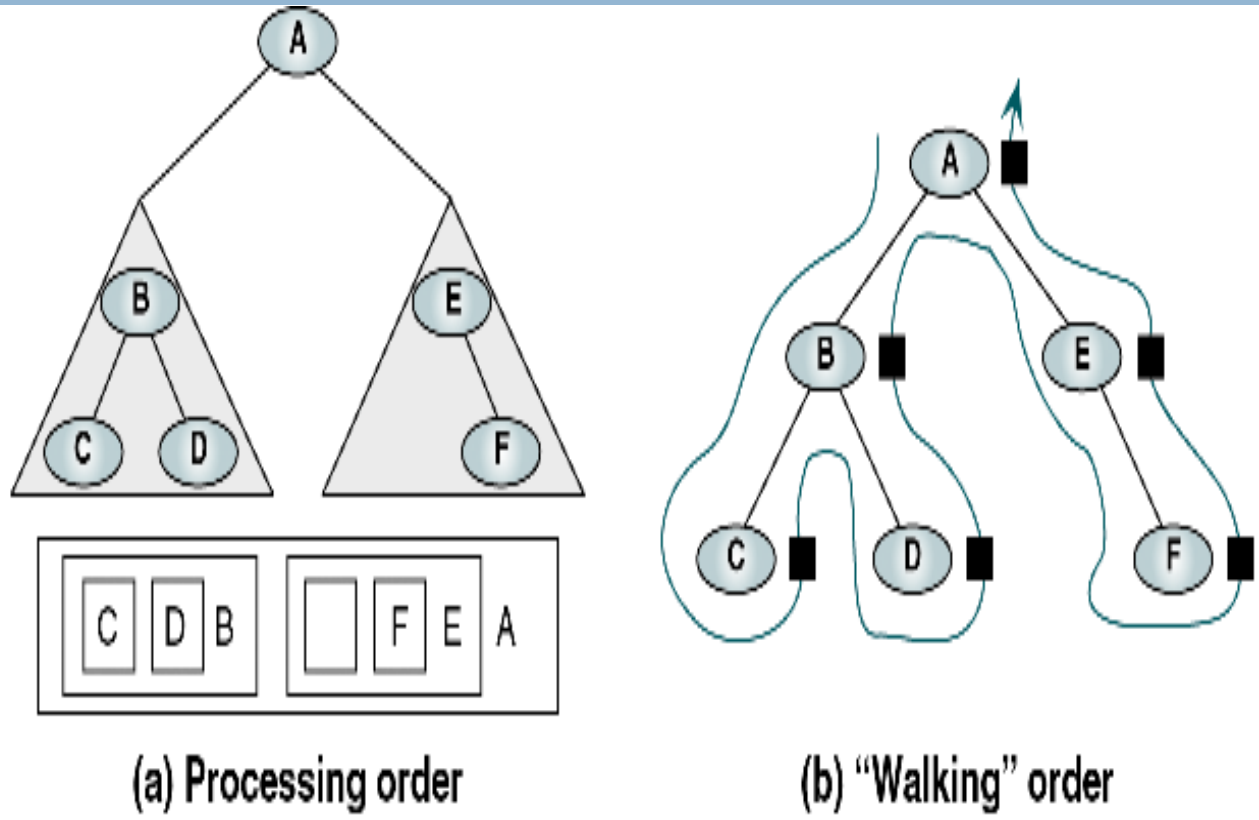
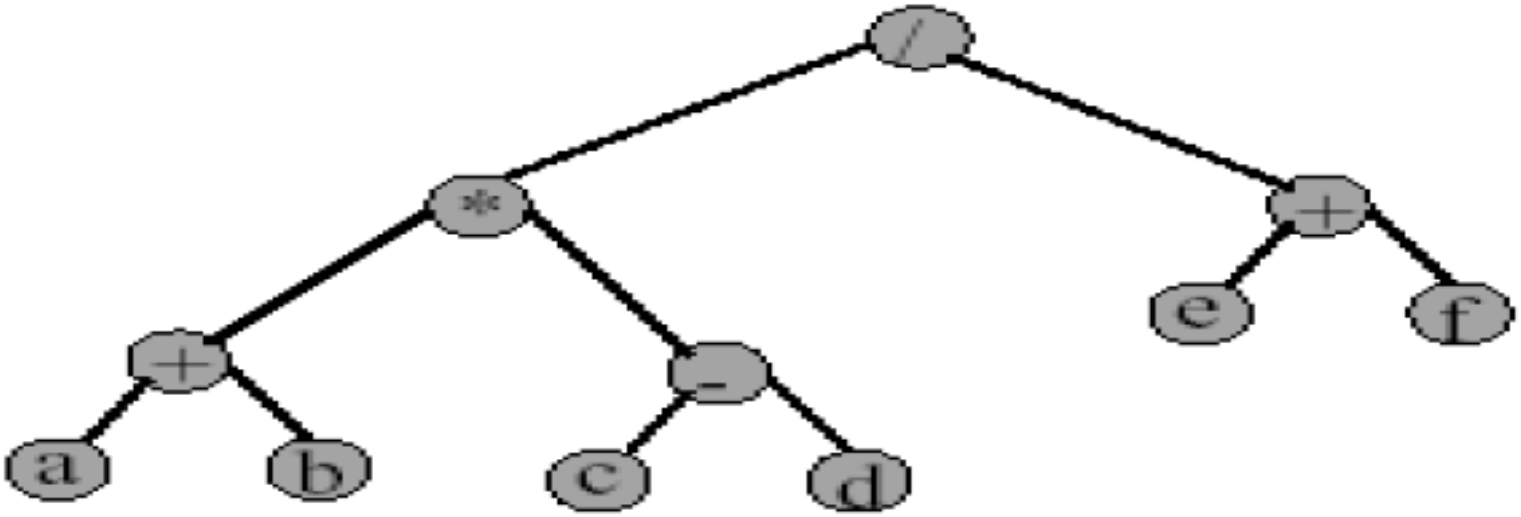


FIGURE 6-13 Postorder Traversal—C D B F E A

Postorder of Expression Tree



a b + c d - * e f + /

Gives postfix form of expression.

ALGORITHM 6-5 Breadth-first Tree Traversal

```
Algorithm breadthFirst (root)
Process tree using breadth-first traversal.
   Pre    root is node to be processed
   Post   tree has been processed
 1 set currentNode to root
 2 createQueue (bfQueue)
 3 loop (currentNode not null)
   1  process (currentNode)
   2  if (left subtree not null)
      1  enqueue (bfQueue, left subtree)
   3  end if
   4  if (right subtree not null)
      1  enqueue (bfQueue, right subtree)
   5  end if
   6  if (not emptyQueue(bfQueue))
      1  set currentNode to dequeue (bfQueue)
   7  else
      1  set currentNode to null
   8  end if
 4 end loop
 5 destroyQueue (bfQueue)
end breadthFirst
```

Breadth-first Traversal

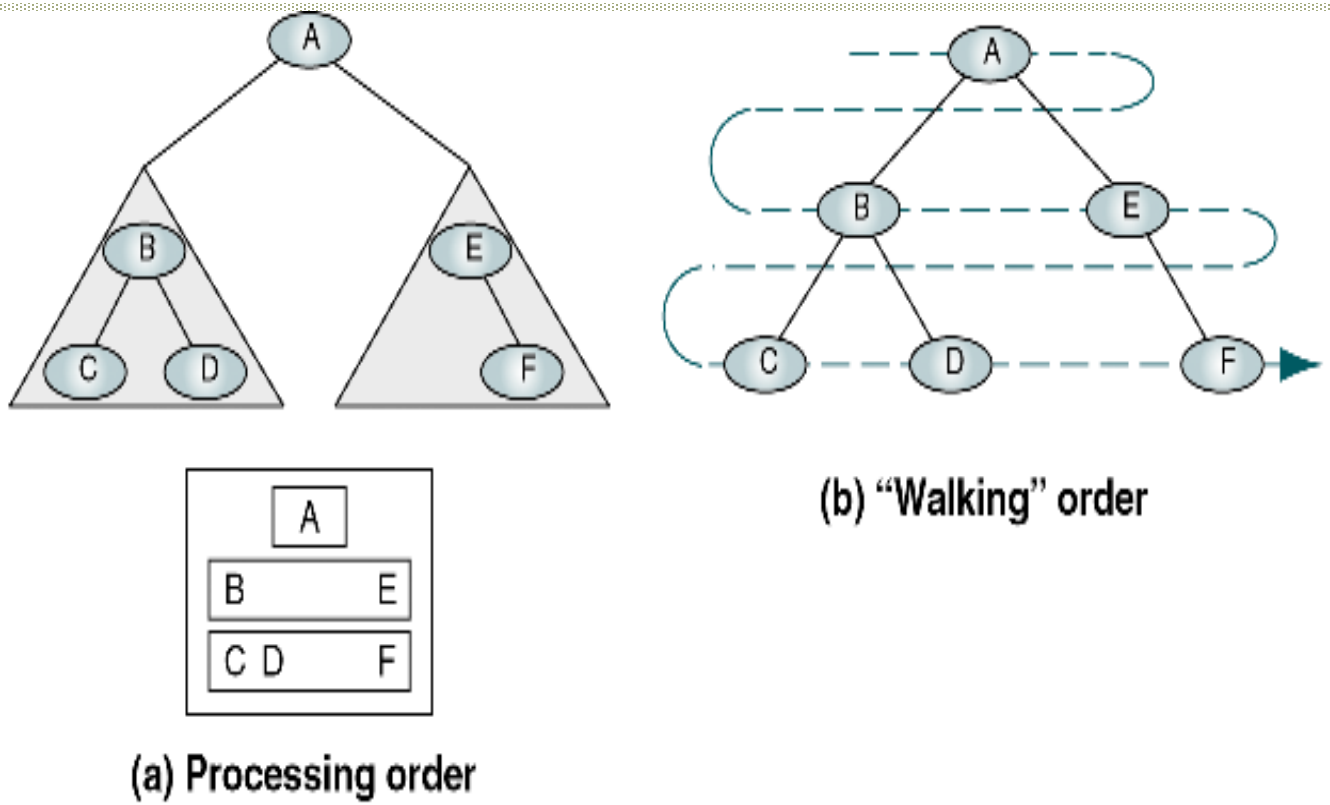
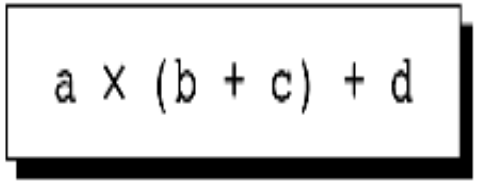


FIGURE 6-14 Breadth-first Traversal

Infix Expression and Its Expression Tree



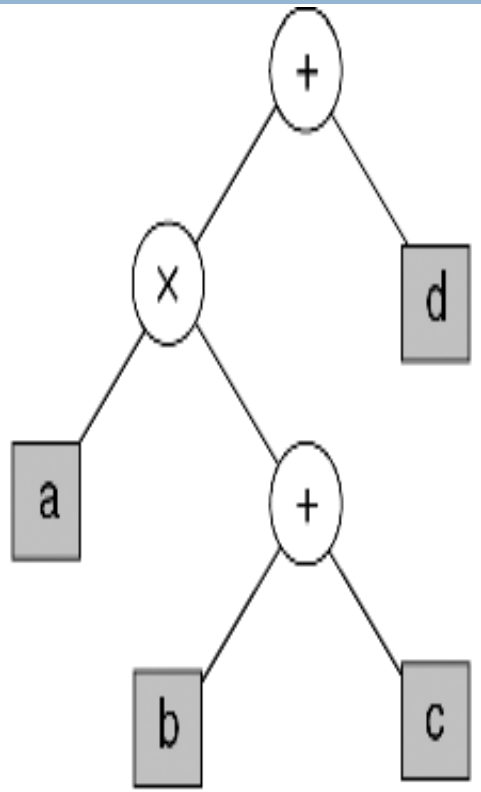
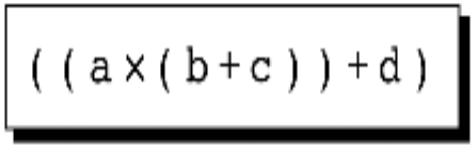


FIGURE 6-15 Infix Expression and Its Expression Tree

Infix Traversal of an Expression Tree



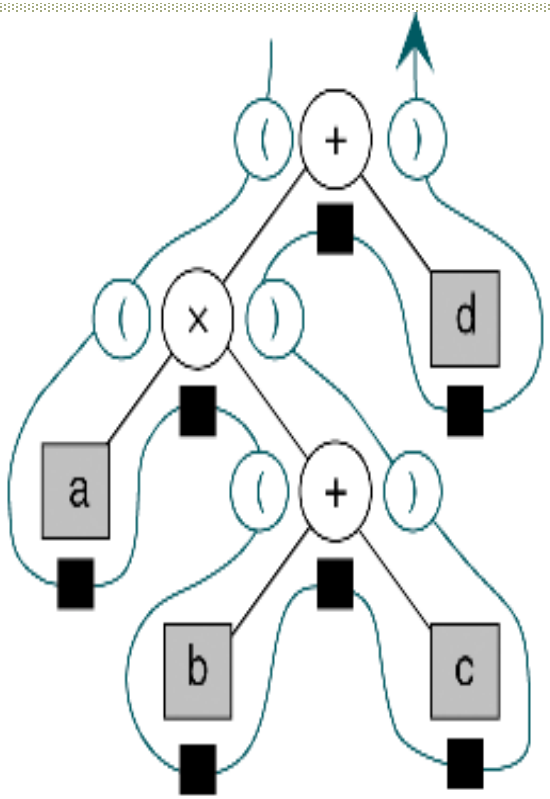


FIGURE 6-16 Infix Traversal of an Expression Tree

ALGORITHM 6-6 Infix Expression Tree Traversal


```
Algorithm infix (tree)
Print the infix expression for an expression tree.
   Pre  tree is a pointer to an expression tree
   Post the infix expression has been printed
1  if (tree not empty)
   1   if (tree token is an operand)
       1   print (tree-token)
   2   else
       1   print (open parenthesis)
       2   infix (tree left subtree)
       3   print (tree token)
       4   infix (tree right subtree)
       5   print (close parenthesis)
   3   end if
2  end if
end infix
```

ALGORITHM 6-7 Postfix Traversal of an Expression Tree

```
Algorithm postfix (tree)
Print the postfix expression for an expression tree.
   Pre  tree is a pointer to an expression tree
   Post the postfix expression has been printed
 1 if (tree not empty)
      1  postfix (tree left subtree)
      2  postfix (tree right subtree)
      3  print   (tree token)
 2 end if
end postfix
```

ALGORITHM 6-8 Prefix Traversal of an Expression Tree


```
Algorithm prefix (tree)
Print the prefix expression for an expression tree.
   Pre  tree is a pointer to an expression tree
   Post the prefix expression has been printed
1  if (tree not empty)
   1  print  (tree token)
   2  prefix (tree left subtree)
   3  prefix (tree right subtree)
2  end if
end prefix
```

Space and Time Complexity

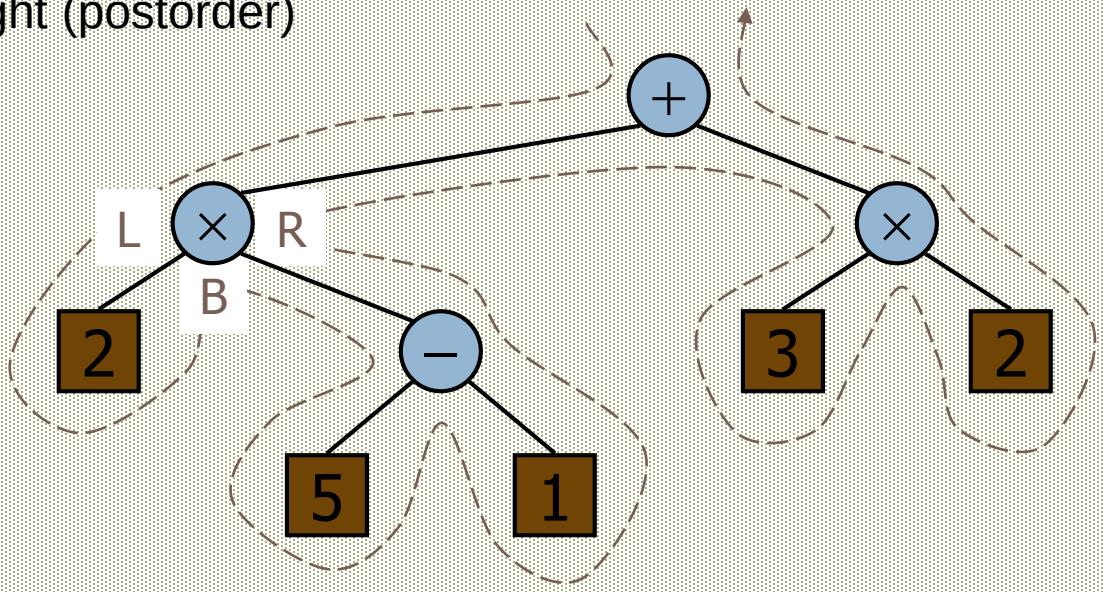
- The space complexity of each of the four traversal algorithm is **O(n)**
- The time complexity of each of the four traversal algorithm is **O(n)**

Euler Tour Traversal

- **Euler tour** is the path along the tree that begins at the root and ends at the root, traversing each edge exactly twice — once to enter the subtree at the other endpoint and once to leave it.
- You can think of an Euler tour as just being a depth first traversal where we return to the root at the end.

Euler Tour Traversal

- Generic traversal of a binary tree
- Includes special cases the preorder, postorder and inorder traversals
- Walk around the tree and visit each node three times:
  - on the left (preorder)
  - from below (inorder)
  - on the right (postorder)



Euler Tour Traversal

```
eulerTour(node v) {
    perform action for visiting node on the left;
    if v is internal then
        eulerTour(v->left);
    perform action for visiting node from below;
    if v is internal then
        eulerTour(v->right);
    perform action for visiting node on the right;
}
```

Binary Search Trees (BST)

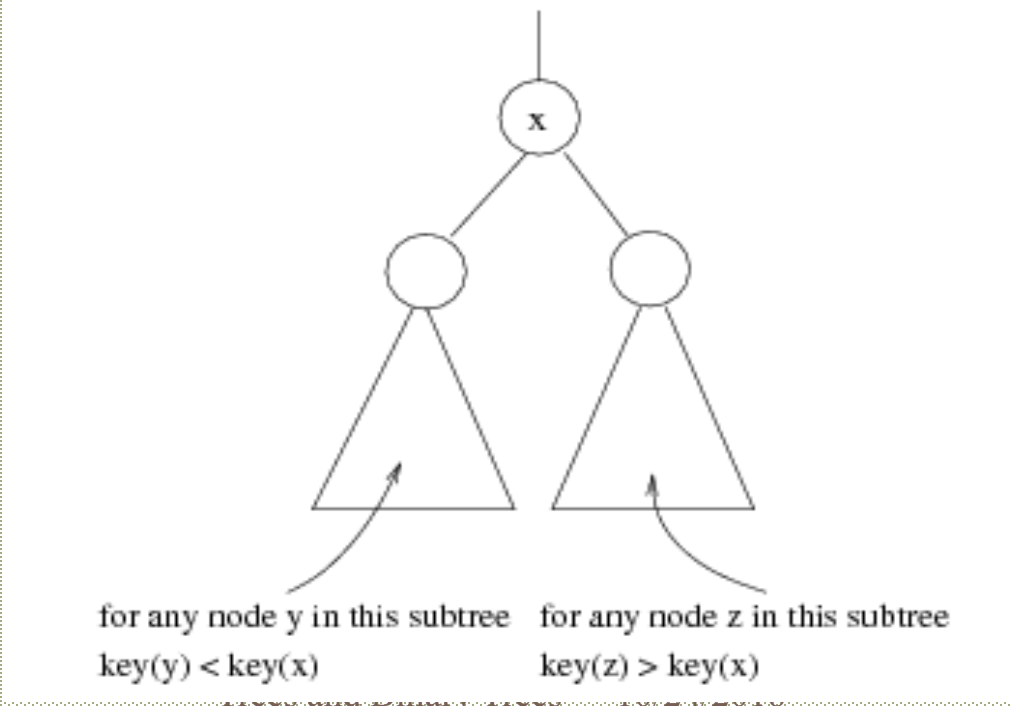
- View today as data structures that can support dynamic set operations.
  - Search, Minimum, Maximum, Predecessor, Successor, Insert, and Delete.
- Can be used to build
  - » Dictionaries.
  - » Priority Queues.
- Basic operations take time proportional to the height of the tree – $O(h)$.

Binary Search Trees (BST)

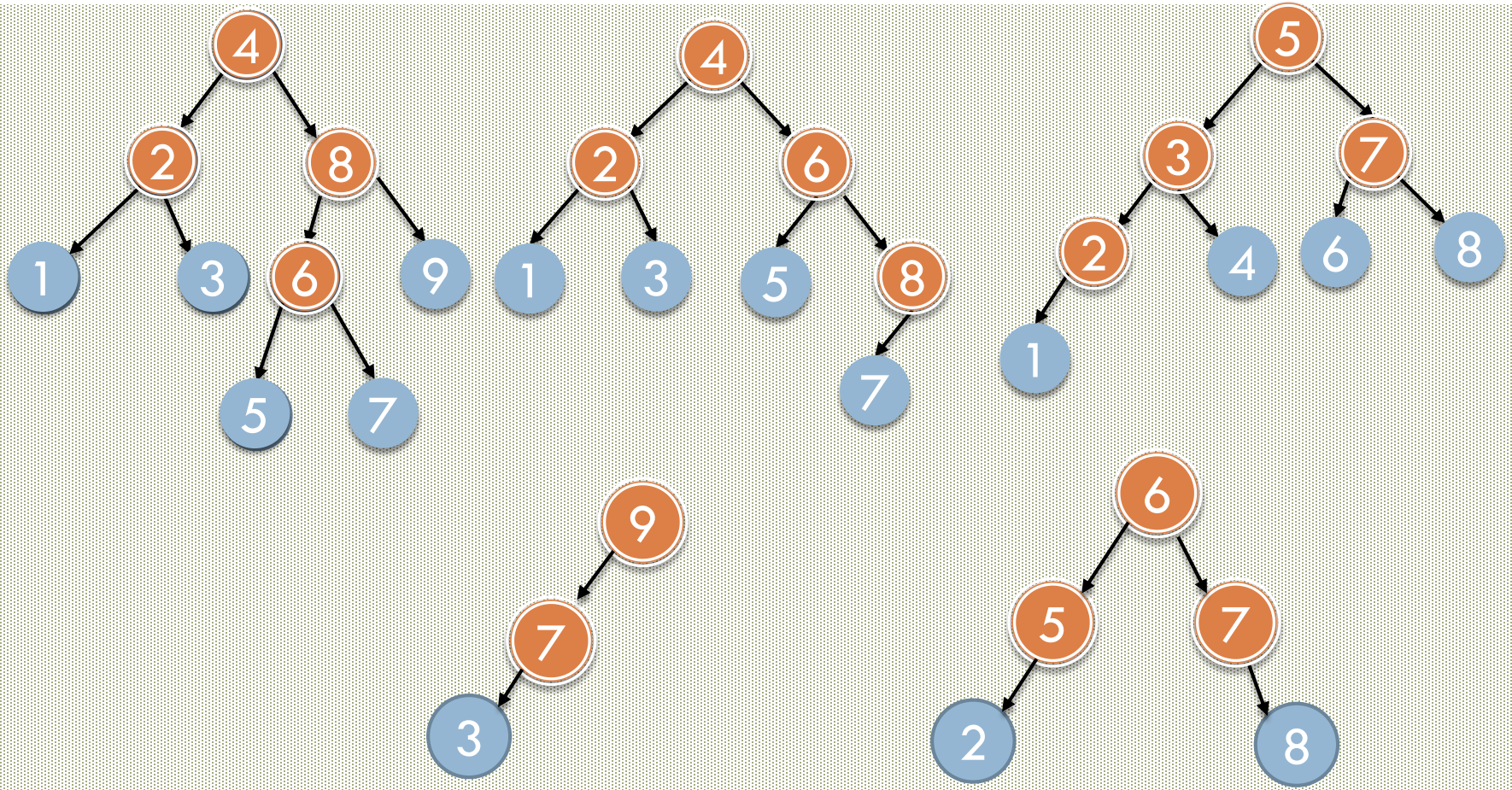
- A binary tree in which each node value of all the nodes in left subtree is lesser or equal and value of all the nodes in the right subtree is greater.
- Stores keys in the nodes in a way so that searching, insertion and deletion can be done efficiently.

Binary search tree property

- For every node X, all the keys in its left subtree are smaller than the key value in X, and all the keys in its right subtree are larger than the key value in X

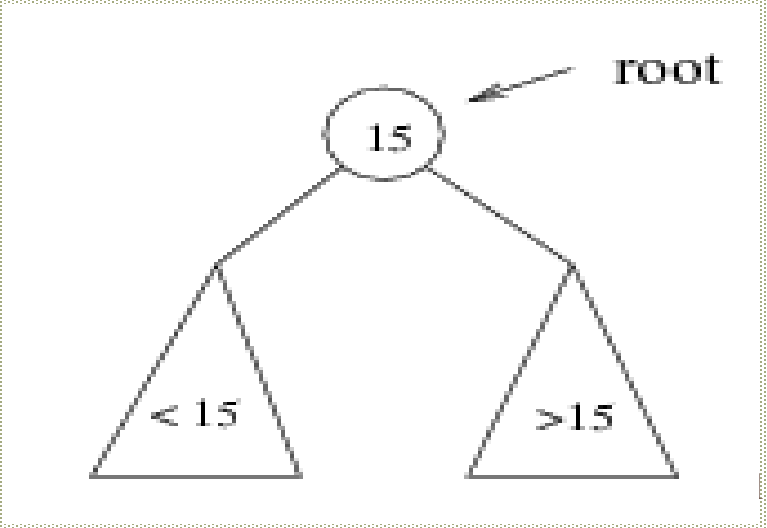


Binary Search Trees Examples

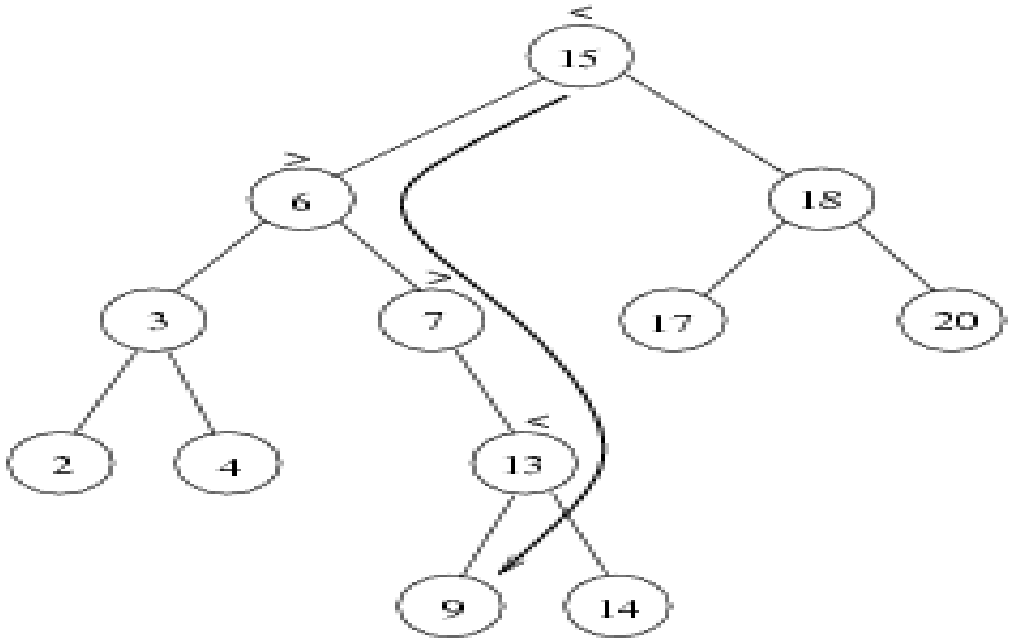


Searching BST

- If we are searching for 15, then we are done.
- If we are searching for a key < 15, then we should search in the left subtree.
- If we are searching for a key > 15, then we should search in the right subtree.



Example: Search for 9 ...

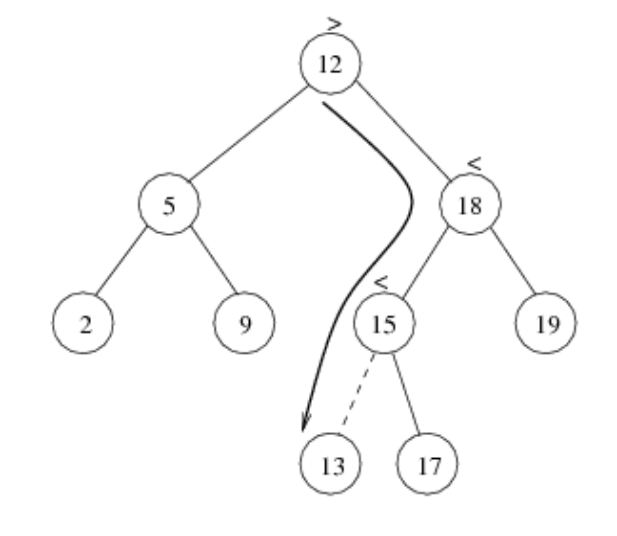


Search for 9:

1. compare 9:15(the root), go to left subtree;
2. compare 9:6, go to right subtree;
3. compare 9:7, go to right subtree;
4. compare 9:13, go to left subtree;
5. compare 9:9, found it!

Insert

- Proceed down the tree as you would with a find
- If X is found, do nothing (or update something)
- Otherwise, insert X at the last spot on the path traversed



- ● Time complexity = O(height of the tree)

Delete

- When we delete a node, we need to consider how we take care of the children of the deleted node.
  - This has to be done such that the property of the search tree is maintained.

Delete

- Three cases:

  (1) the node is a leaf

  – Delete it immediately

  (2) the node has one child

  – Adjust a pointer from the parent to bypass that node

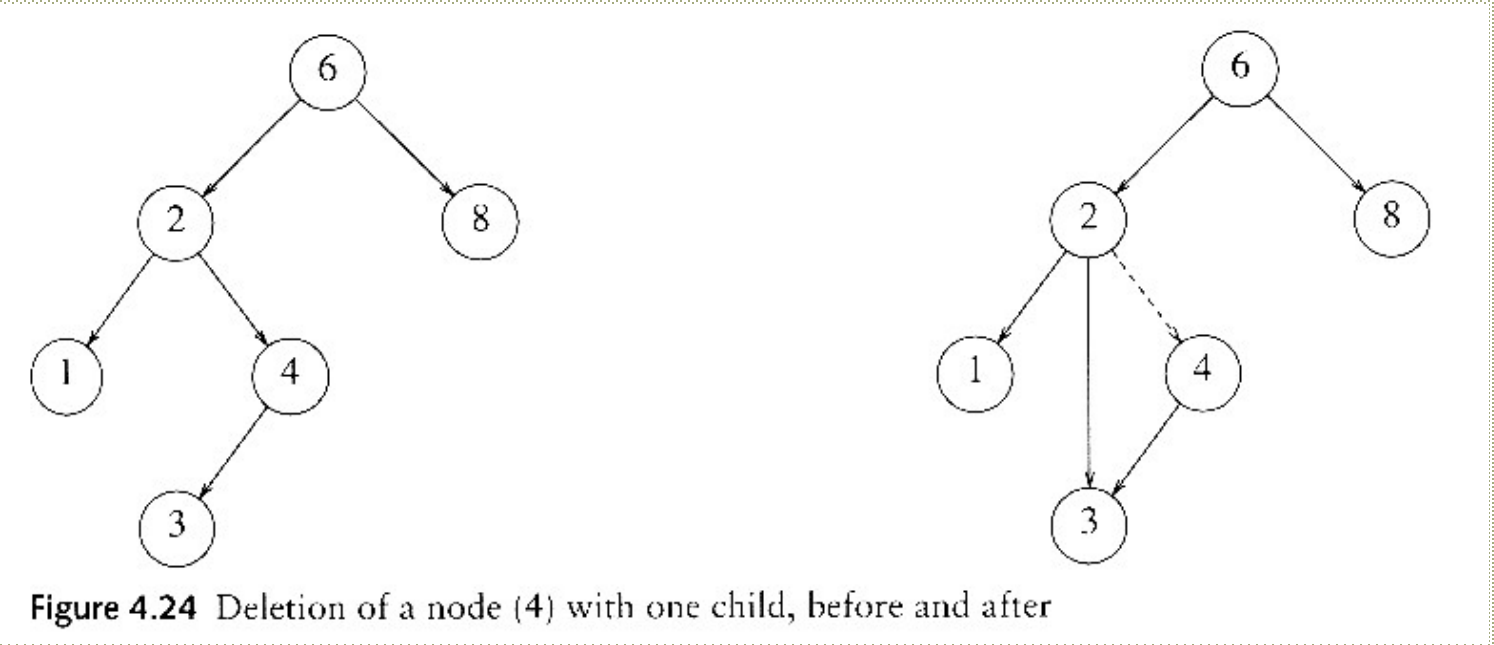


Figure 4.24 Deletion of a node (4) with one child, before and after

The End