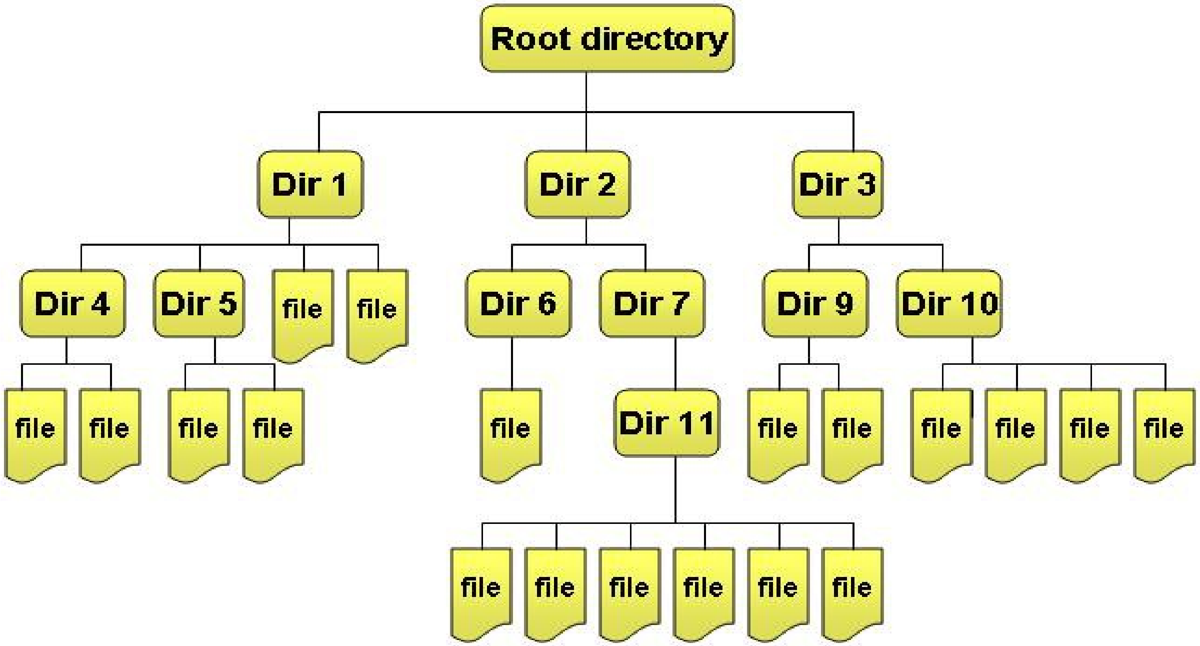
TREES

- In this session, you will learn to:
  - Store data in a tree
  - Distinguish types of Binary tree
  - Traverse a Binary Tree
    - InOrder
    - PreOrder
    - PostOrder
  - Construct a Binary Tree
  - Construct an expression Binary tree

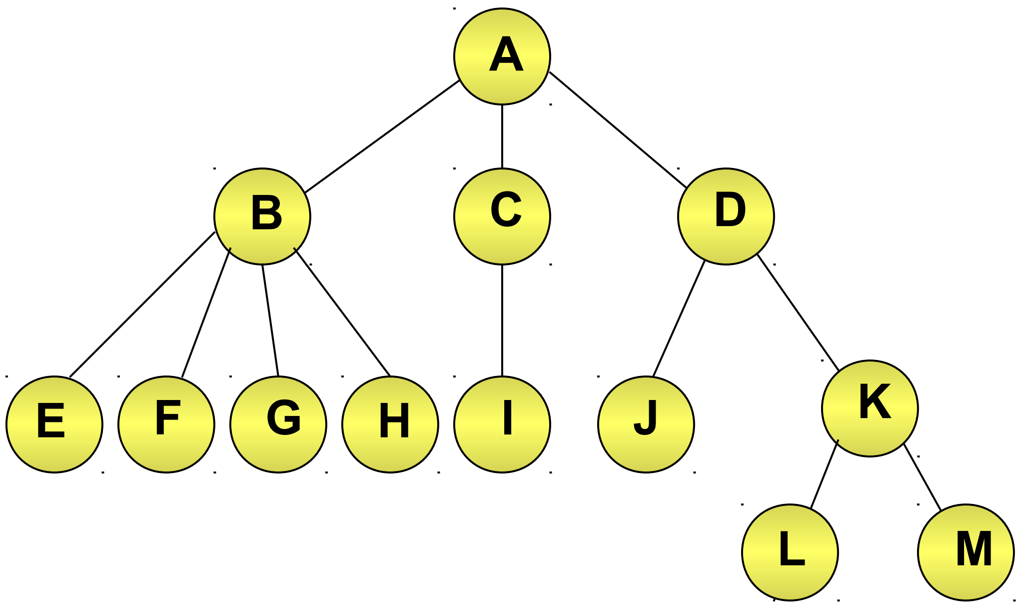
Storing Data in a Tree

- Consider a scenario where you are required to represent the directory structure of your operating system.
- The directory structure contains various folders and files. A folder may further contain any number of sub folders and files.
- In such a case, it is not possible to represent the structure linearly because all the items have a hierarchical relationship among themselves.
- In such a case, it would be good if you have a data structure that enables you to store your data in a nonlinear fashion.
- Directory structure:



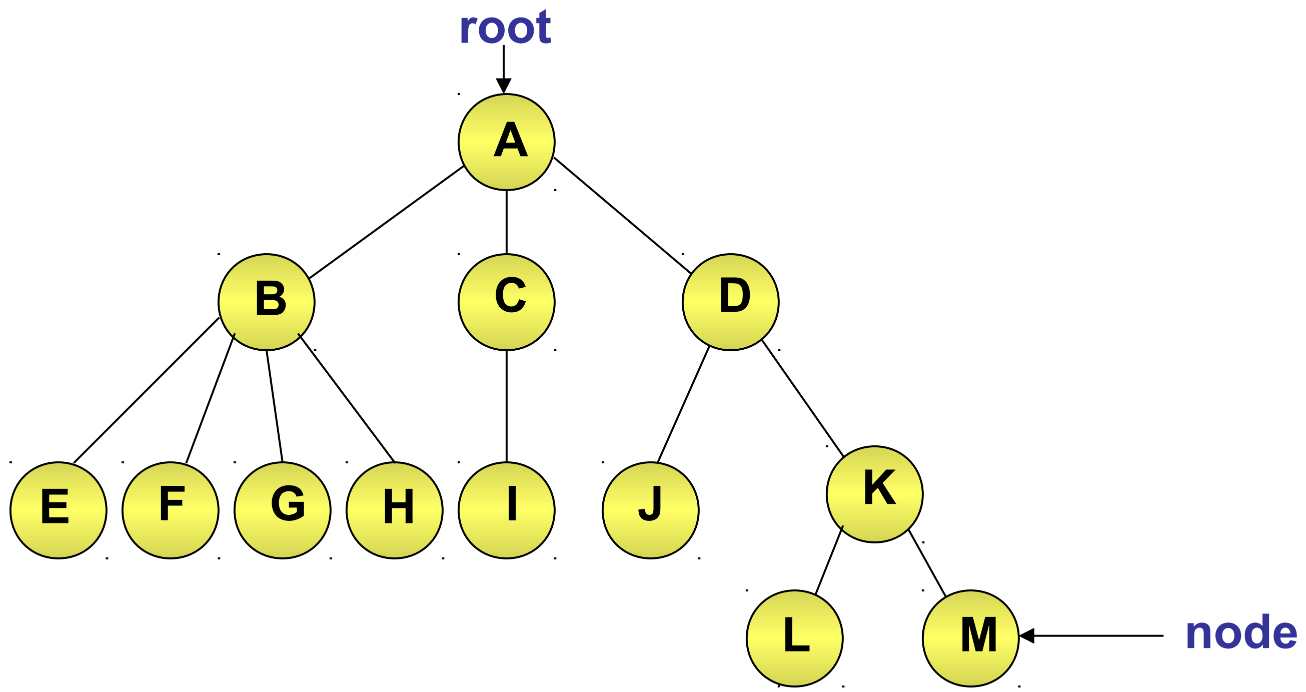
Defining Trees

- A tree is a nonlinear data structure that represent a hierarchical relationship among the various data elements.
- Trees are used in applications in which the relation between data elements needs to be represented in a hierarchy.



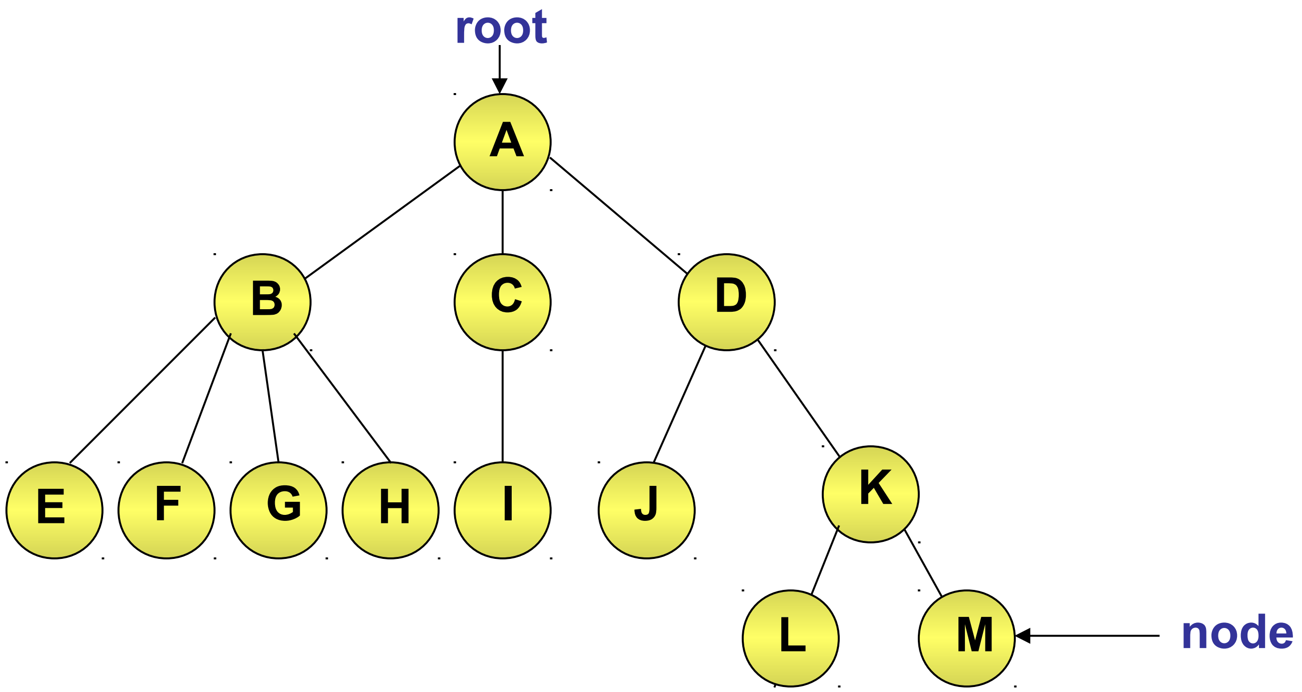
Defining Trees (Contd.)

- Each element in a tree is referred to as a node.
- The topmost node in a tree is called root.



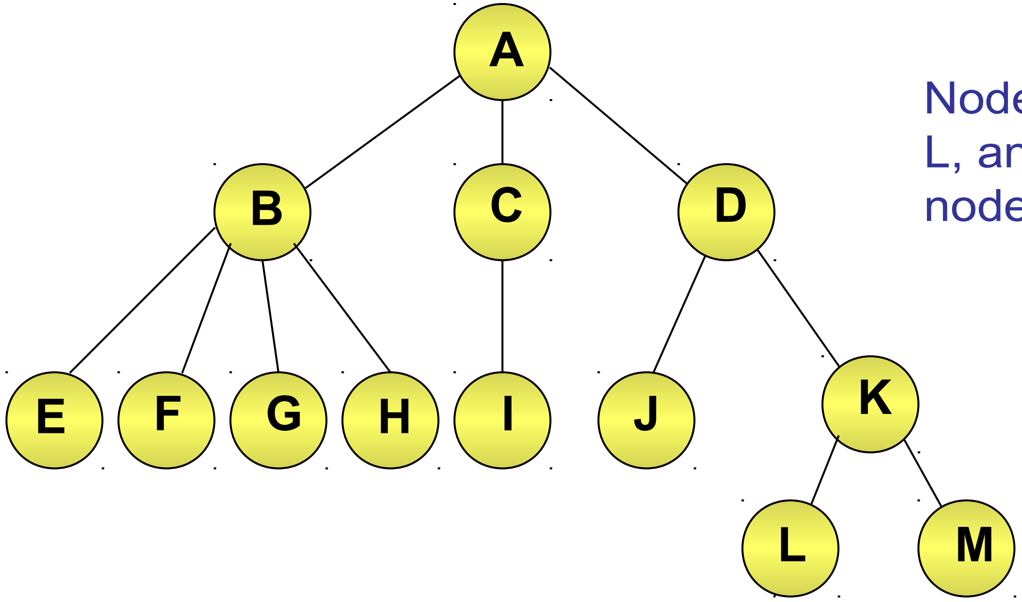
Defining Trees (Contd.)

- Each node in a tree can further have subtrees below its hierarchy.



Tree Terminology

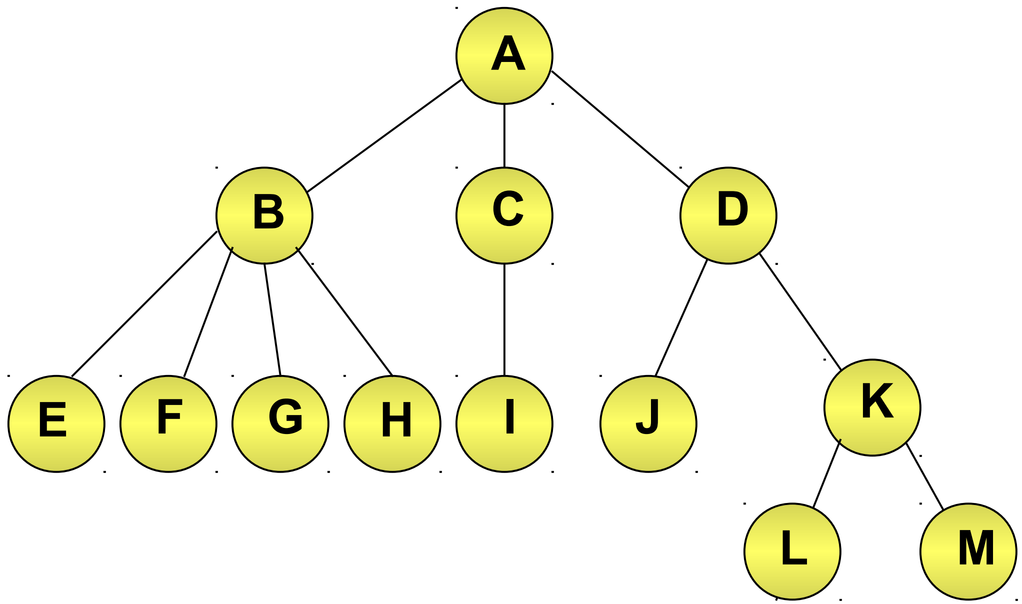
- Let us discuss various terms that are most frequently used with trees.
  - **Leaf node**: It refers to a node with no children.



Nodes E, F, G, H, I, J, L, and M are leaf nodes.

Tree Terminology (Contd.)

- **Subtree**: A portion of a tree, which can be viewed as a separate tree in itself is called a subtree.
  - A subtree can also contain just one node called the root node.
- **Children of a node**: The roots of the subtrees of a node are called the children of the node.

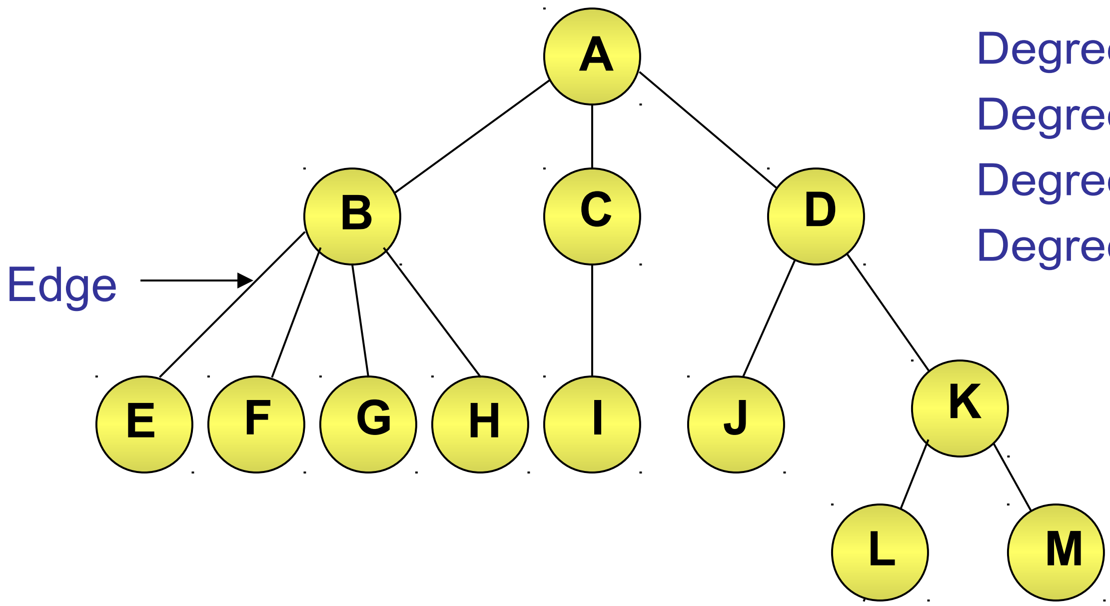


Tree with root B, containing nodes E, F, G, and H is a subtree of node A.

E, F, G, and H are children of node B. B is the parent of these nodes.

Tree Terminology (Contd.)

- **Edge**: A link from the parent to a child node is referred to as an edge.
- **Degree of a node**: It refers to the number of subtrees of a node in a tree.



Degree of node C is 1

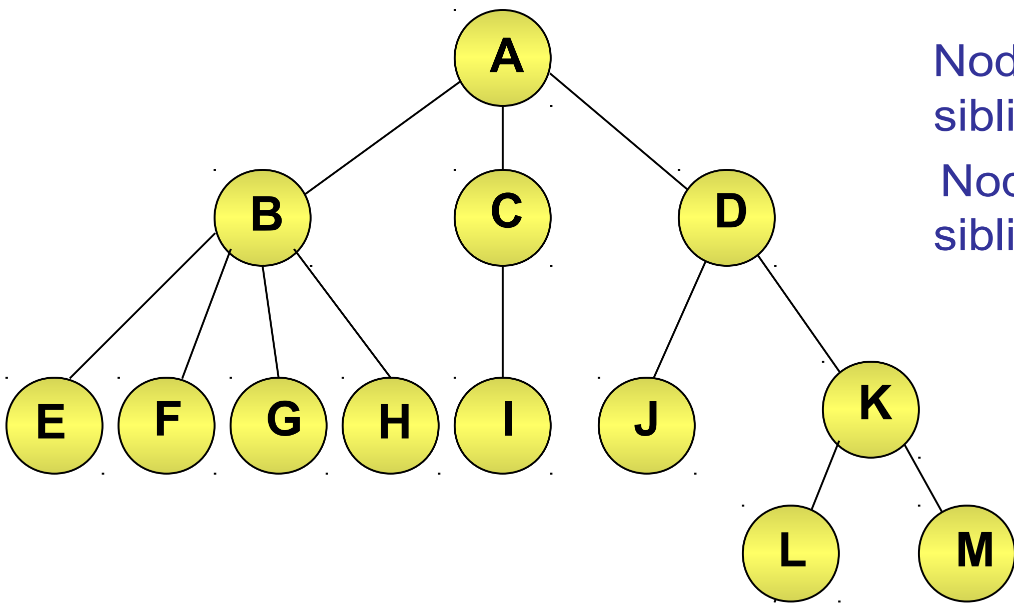
Degree of node D is 2

Degree of node A is 3

Degree of node B is 4

Tree Terminology (Contd.)

- **Siblings/Brothers**: It refers to the children of the same node.

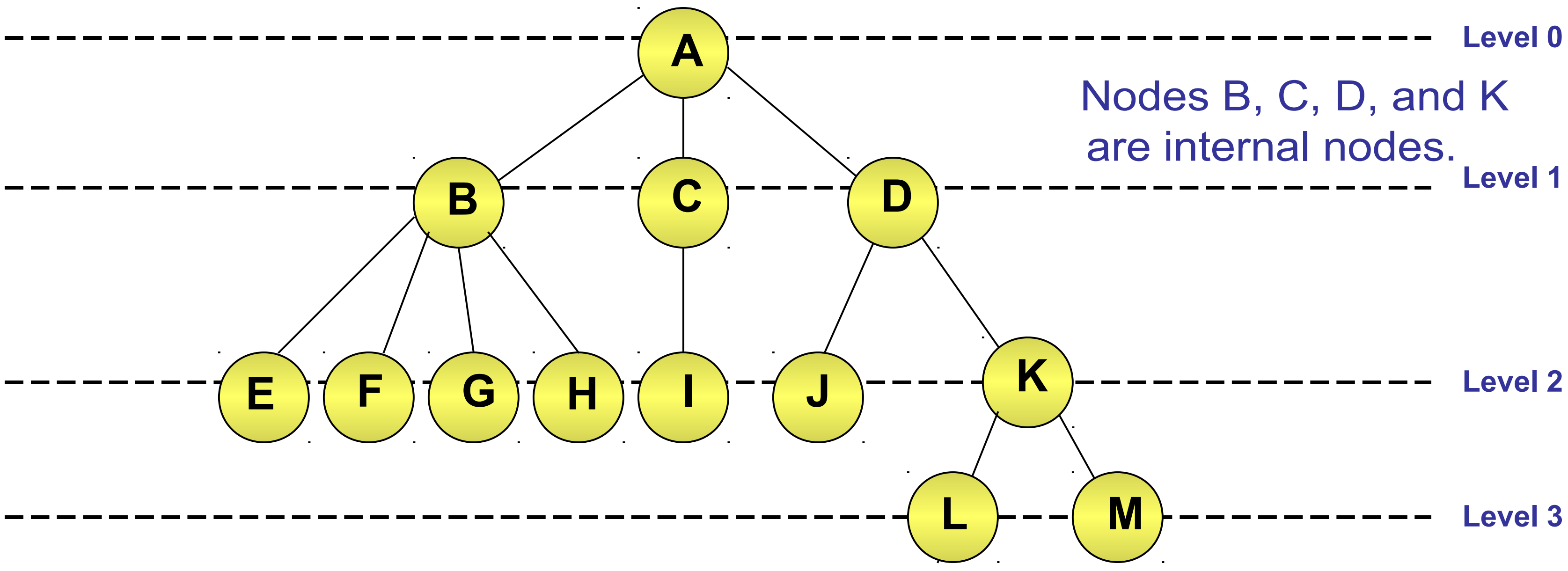


Nodes B, C, and D are siblings of each other.

Nodes E, F, G, and H are siblings of each other.

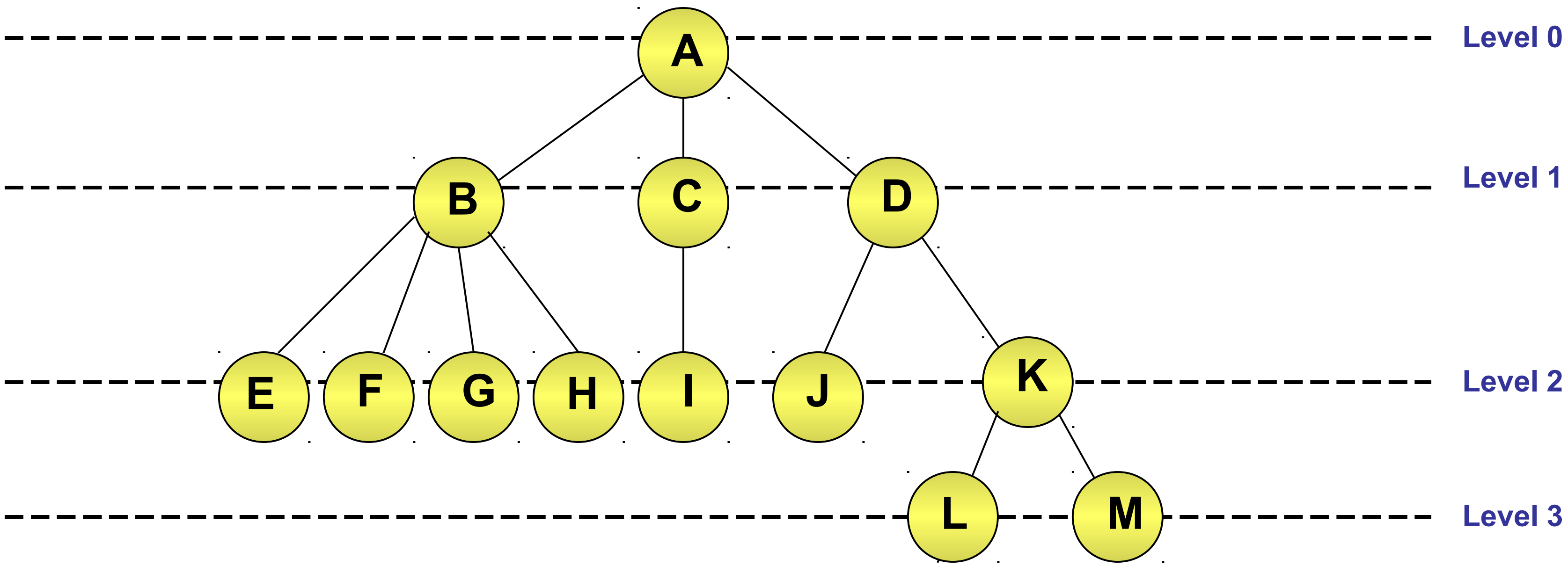
Tree Terminology (Contd.)

- **Internal node**: It refers to any node between the root and a leaf node.
- **Level of a node**: It refers to the distance (in number of nodes) of a node from the root. Root always lies at level 0.
  - As you move down the tree, the level increases by one.



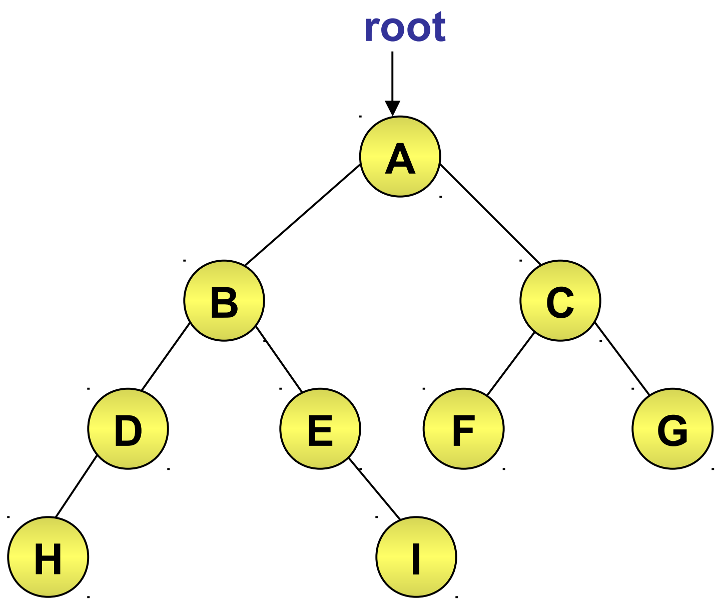
Tree Terminology (Contd.)

- **Depth of a tree**: Refers to the total number of levels in the tree.
  - The depth of the following tree is 4.



Just a minute

- Consider the following tree and answer the questions that follow:
  a. What is the depth of the tree?
  b. Which nodes are children of node B?
  c. Which node is the parent of node F?
  d. What is the level of node E?
  e. Which nodes are the siblings of node H?
  f. Which nodes are the siblings of node D?
  g. Which nodes are leaf nodes?

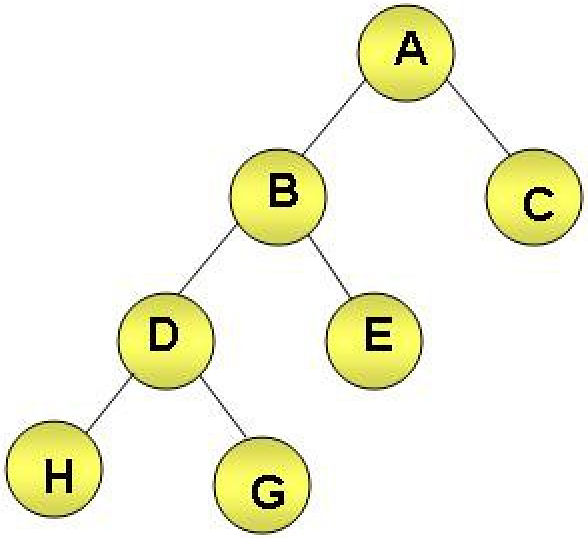


- Answer:
  a. 4
  b. D and E
  c. C
  d. 2
  e. H does not have any siblings
  f. The only sibling of D is E
  g. F, G, H, and I

Defining Binary Trees

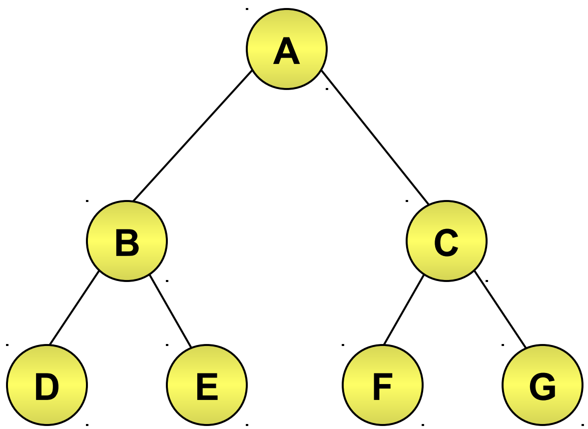
- Binary tree is a specific type of tree in which each node can have at most two children namely left child and right child.
- There are various types of binary trees:
  - Strictly binary tree
  - Full binary tree
  - Complete binary tree

- Strictly binary tree:
  - A binary tree in which every node, except for the leaf nodes, has non-empty left and right children.



Defining Binary Trees (Contd.)

- Full binary tree:
  - A binary tree of depth d that contains exactly $2^d - 1$ nodes.

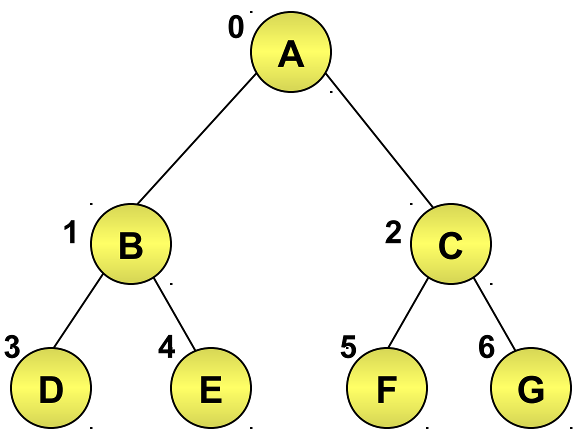


Depth = 3

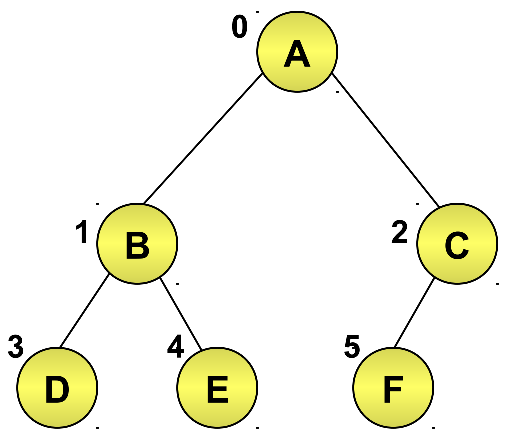
Total number of nodes = $2^3 - 1 = 7$

Defining Binary Trees (Contd.)

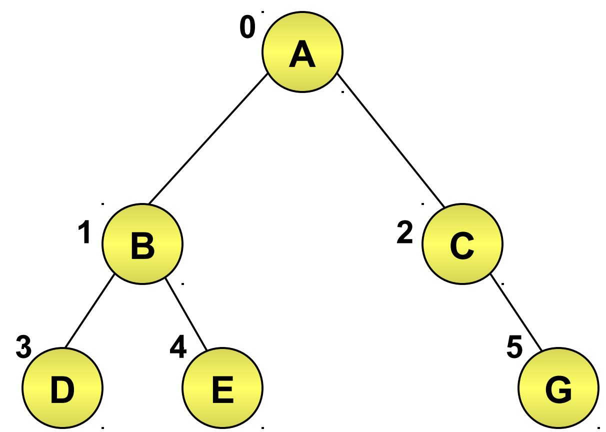
- Complete binary tree:
  - A binary tree with n nodes and depth d whose nodes correspond to the nodes numbered from 0 to n − 1 in the full binary tree of depth k.



Full Binary Tree



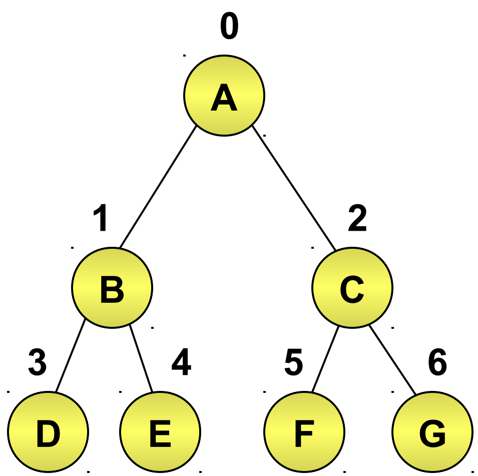
Complete Binary Tree



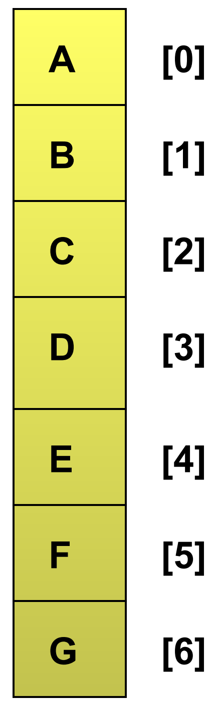
Incomplete Binary Tree

Representing a Binary Tree

- Array representation of binary trees:
  - All the nodes are represented as the elements of an array.



Binary Tree



Array Representation

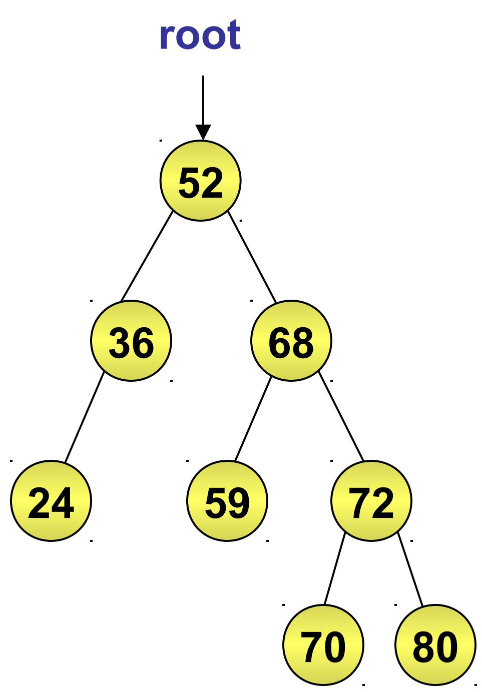
- If there are n nodes in a binary tree, then for any node with index i, where $0 < i < n – 1$:
  - Parent of i is at $(i – 1)/2$.
  - Left child of i is at $2i + 1$:
    - If $2i + 1 > n – 1$, then the node does not have a left child.
  - Right child of i is at $2i + 2$:
    - If $2i + 2 > n – 1$, then the node does have a right child.

Representing a Binary Tree (Contd.)

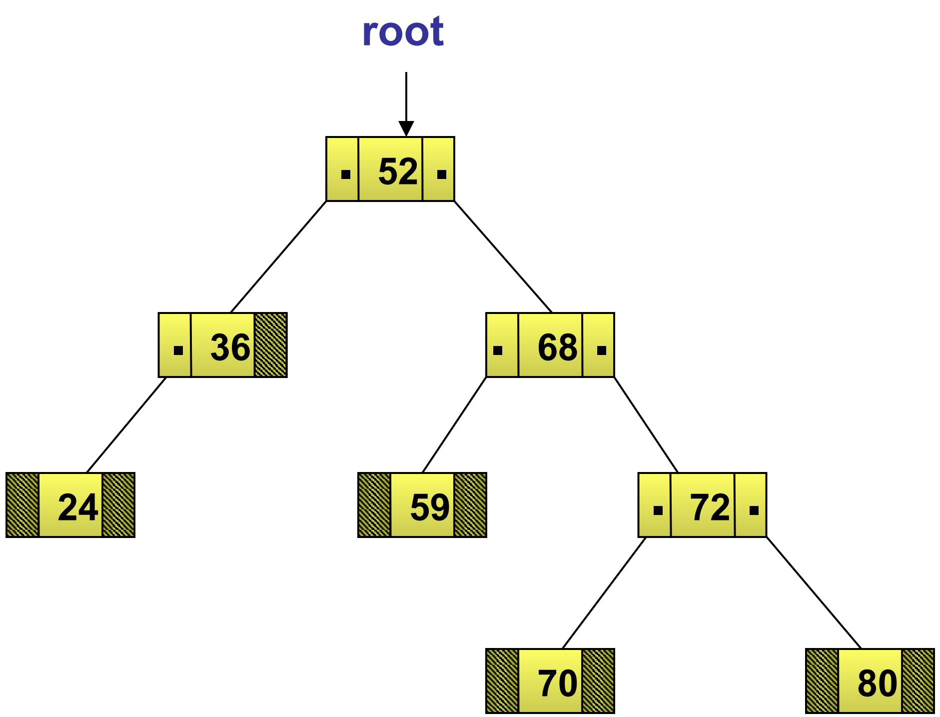
- Linked representation of a binary tree:
  - It uses a linked list to implement a binary tree.
  - Each node in the linked representation holds the following information:
    - Data
    - Reference to the left child
    - Reference to the right child
  - If a node does not have a left child or a right child or both, the respective left or right child fields of that node point to NULL.

Node

Representing a Binary Tree (Contd.)



Binary Tree



Linked Representation

Binary Tree Node

```
Struct node
 {
  Int info;
 Struct node *left,*right;
  }
```

OPERATIONS ON TREES

Traversing a Binary Tree

1)TRAVERSING

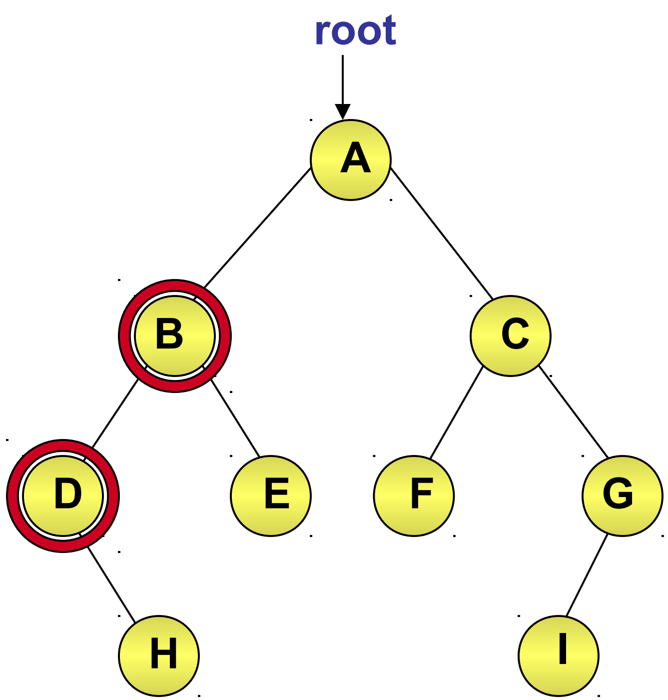
- You can implement various operations on a binary tree.
- A common operation on a binary tree is traversal.
- Traversal refers to the process of visiting all the nodes of a binary tree once.
- There are three ways for traversing a binary tree:
  - Inorder traversal
  - Preorder traversal
  - Postorder traversal

INORDER TRAVERSAL

- Steps for traversing a tree in inorder sequence are as follows:
  1. Traverse the left subtree
  2. Visit root
  3. Traverse the right subtree
- Let us consider an example.

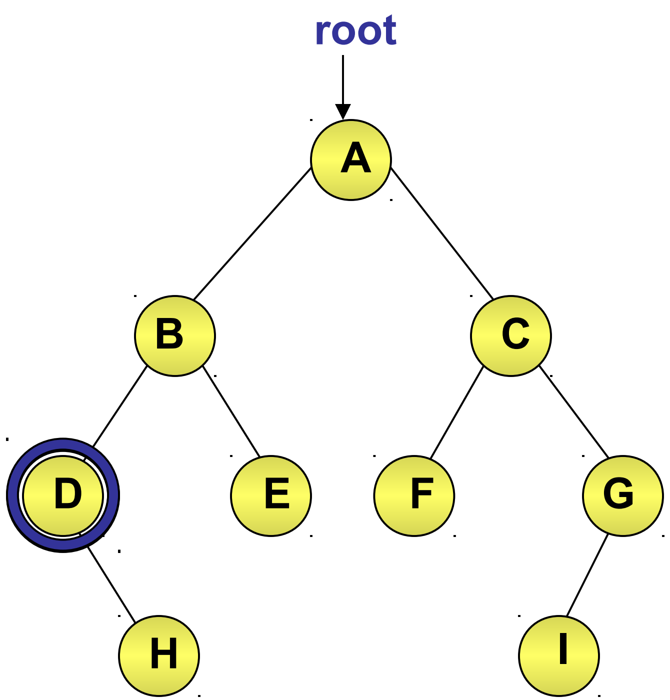
Inorder Traversal (Contd.)

- The left subtree of node B is not NULL.
- Therefore, move to node D to traverse the left subtree of B.



Inorder Traversal (Contd.)

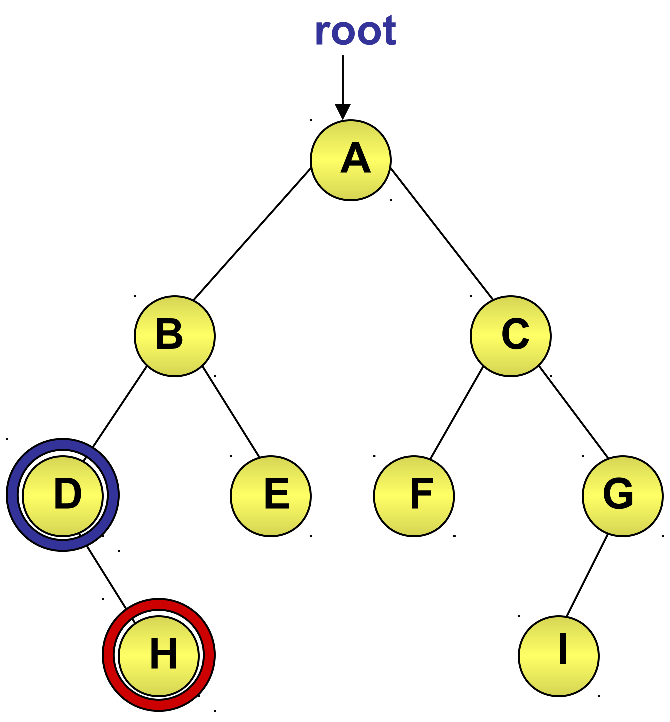
- The left subtree of node D is NULL.
- Therefore, visit node D.



D

Inorder Traversal (Contd.)

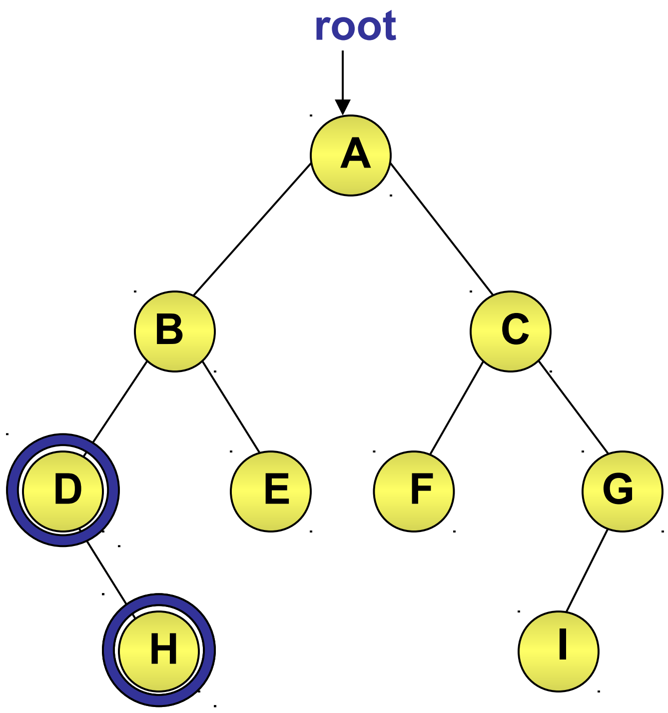
- Right subtree of D is not NULL
- Therefore, move to the right subtree of node D



D

Inorder Traversal (Contd.)

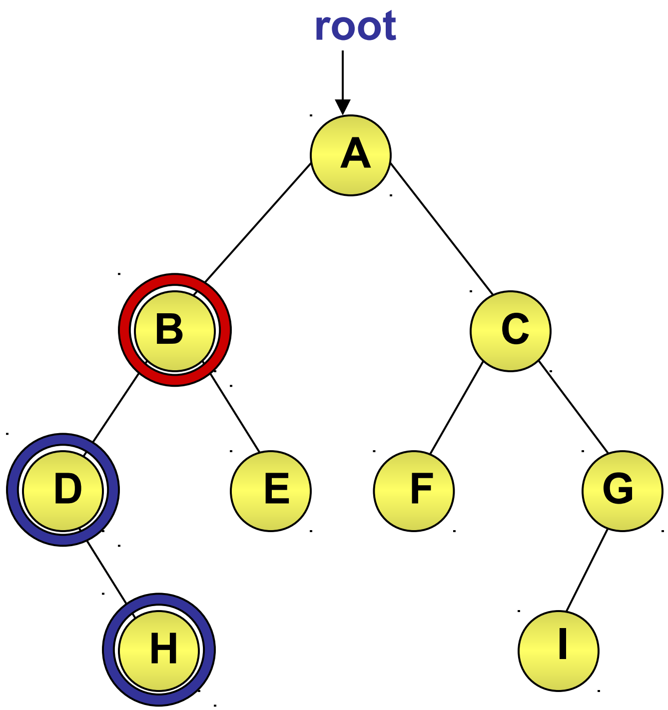
- Left subtree of H is empty.
- Therefore, visit node H.



D H

Inorder Traversal (Contd.)

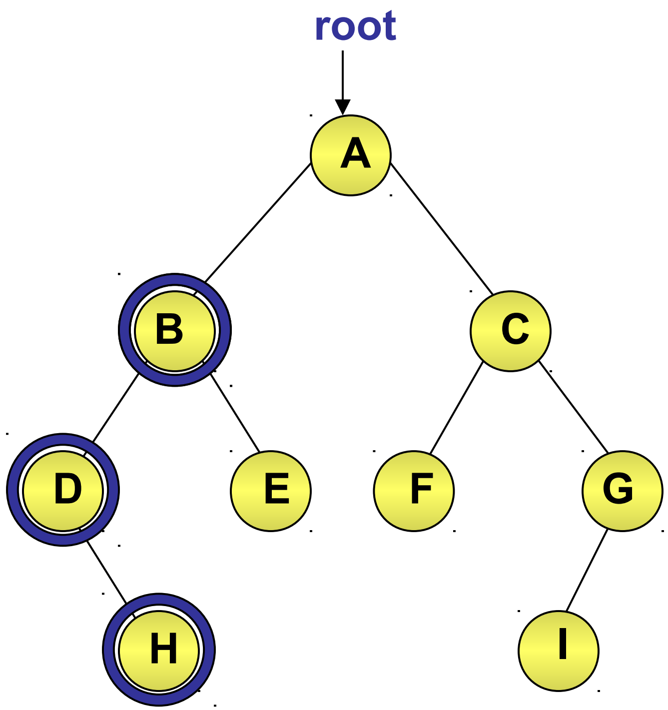
- Right subtree of H is empty.
- Therefore, move to node B.



D H

Inorder Traversal (Contd.)

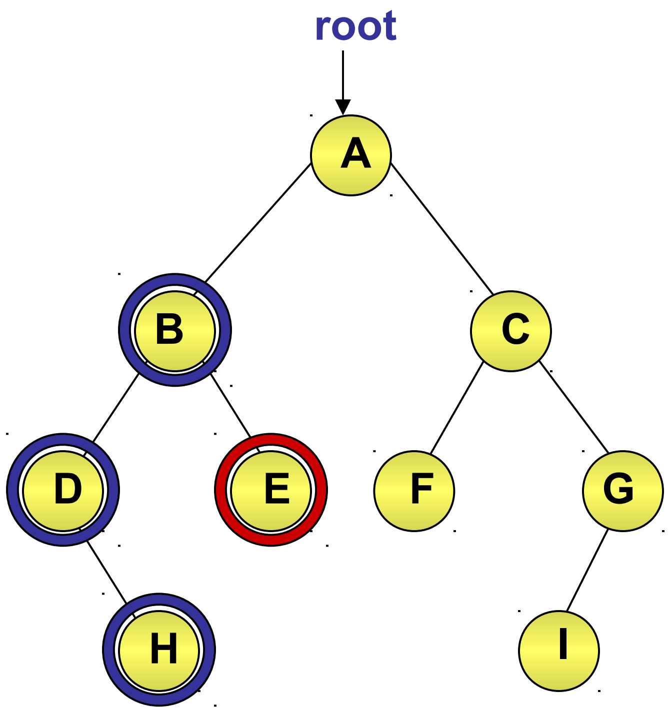
- The left subtree of B has been visited.
- Therefore, visit node B.



D H B

Inorder Traversal (Contd.)

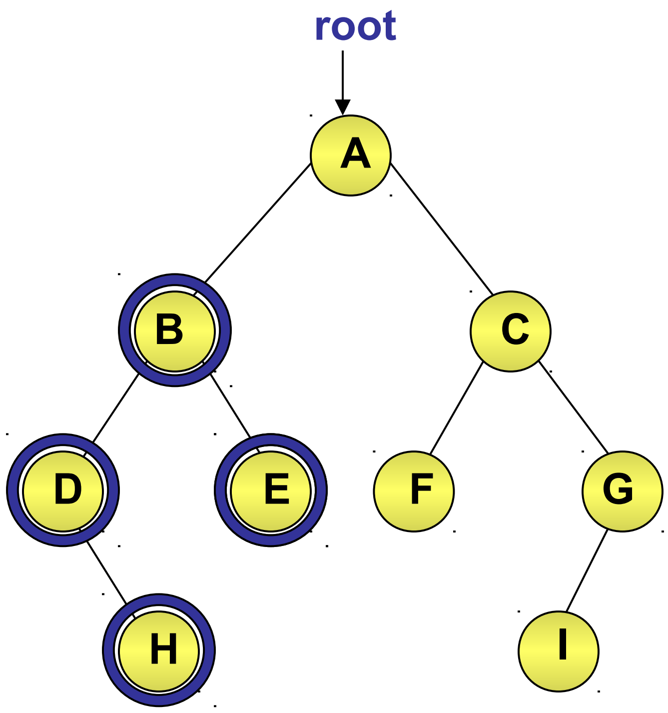
- Right subtree of B is not empty.
- Therefore, move to the right subtree of B.



D H B

Inorder Traversal (Contd.)

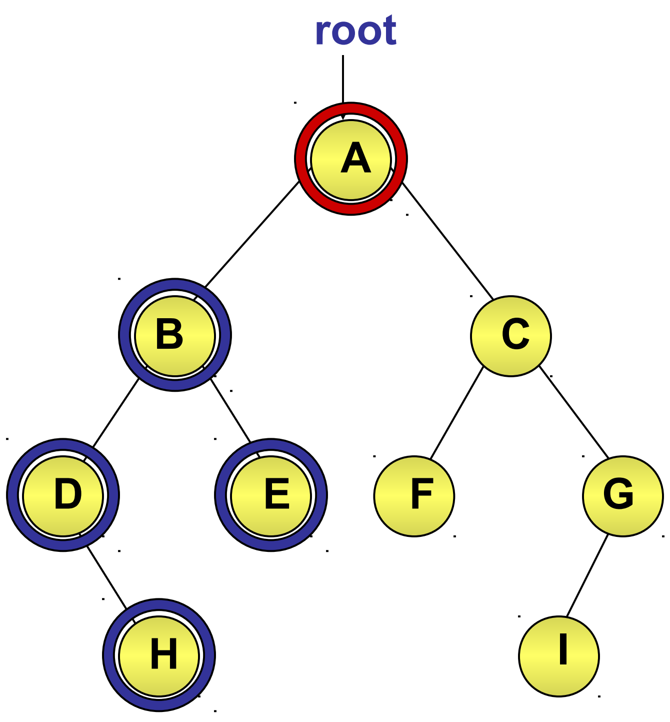
- Left subtree of E is empty.
- Therefore, visit node E.



D H B E

Inorder Traversal (Contd.)

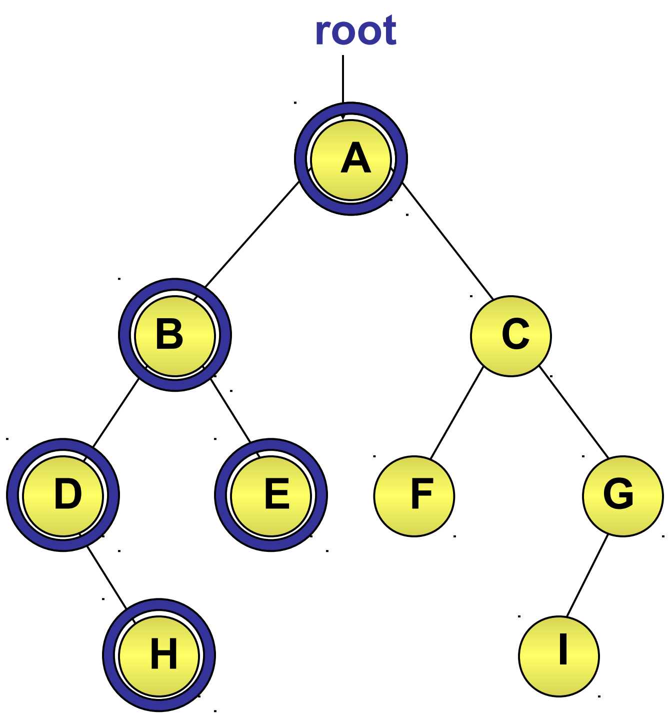
- Right subtree of E is empty.
- Therefore, move to node A.



D H B E

Inorder Traversal (Contd.)

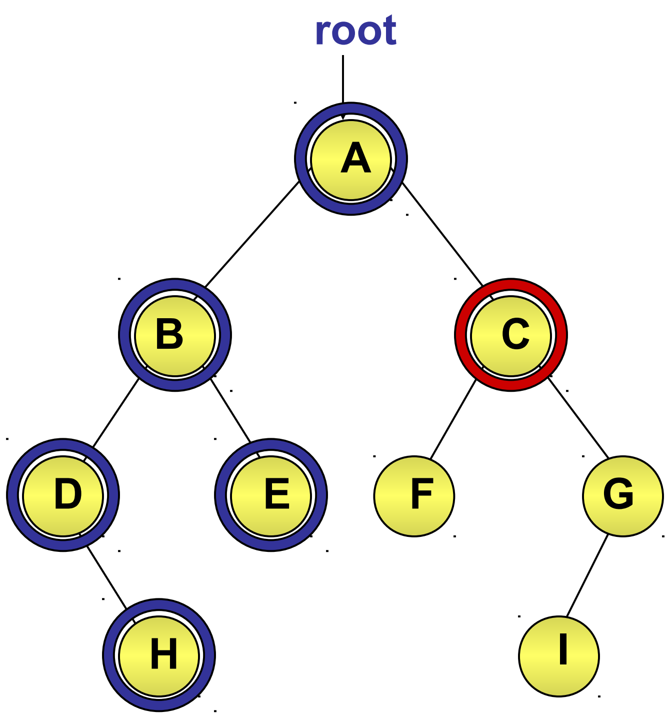
- Left subtree of A has been visited.
- Therefore, visit node A.



D H B E A

Inorder Traversal (Contd.)

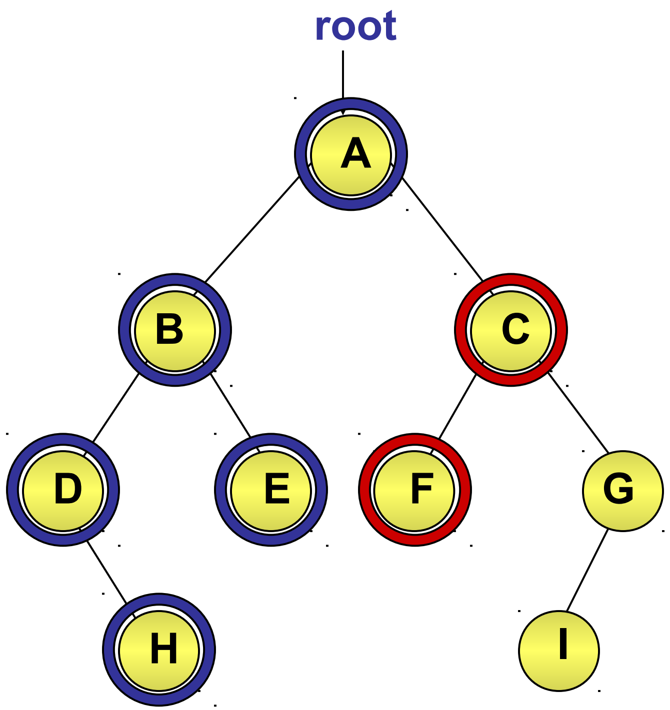
- Right subtree of A is not empty.
- Therefore, move to the right subtree of A.



D H B E A

Inorder Traversal (Contd.)

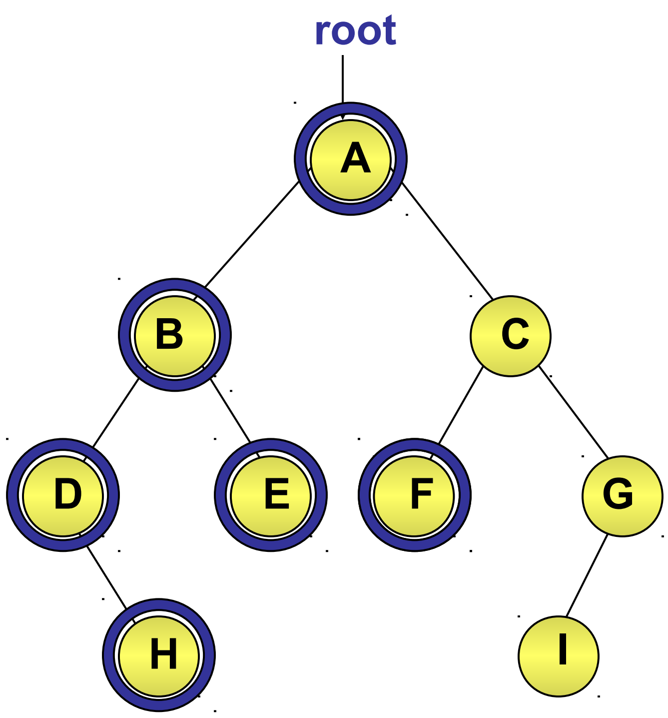
- Left subtree of C is not empty.
- Therefore, move to the left subtree of C.



D H B E A

Inorder Traversal (Contd.)

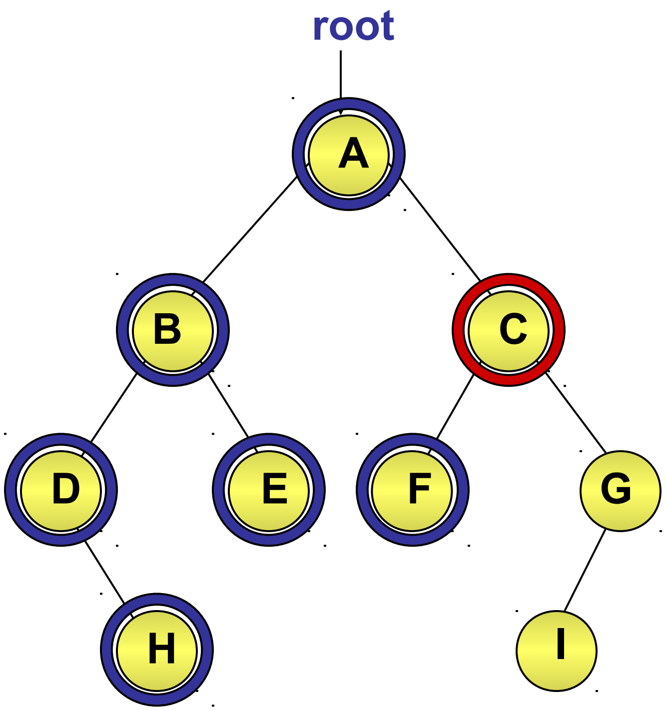
- Left subtree of F is empty.
- Therefore, visit node F.



D H B E A F

Inorder Traversal (Contd.)

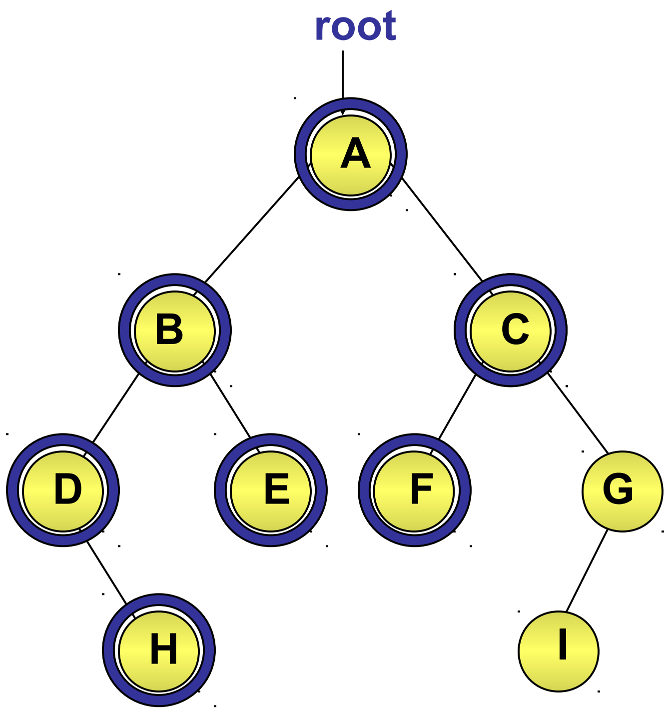
- Right subtree of F is empty.
- Therefore, move to node C.



D H B E A F

Inorder Traversal (Contd.)

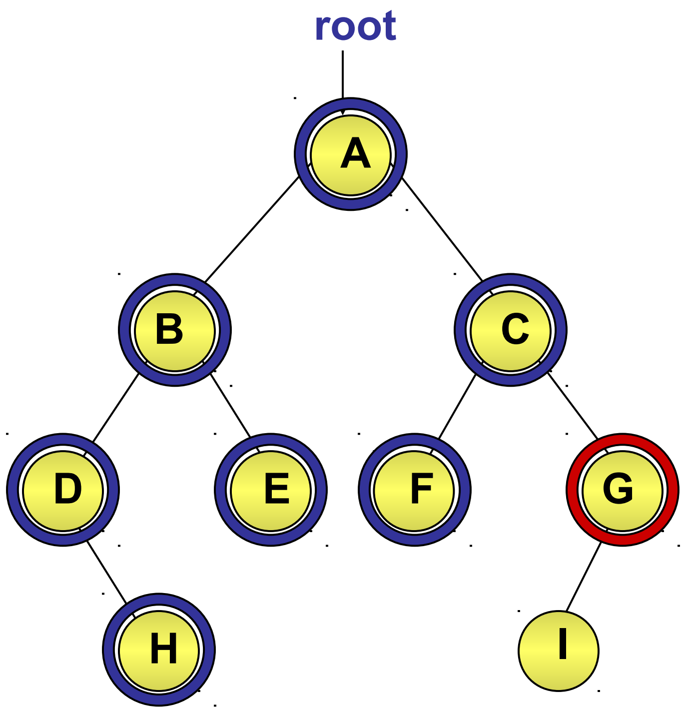
- The left subtree of node C has been visited.
- Therefore, visit node C.



D H B E A F C

Inorder Traversal (Contd.)

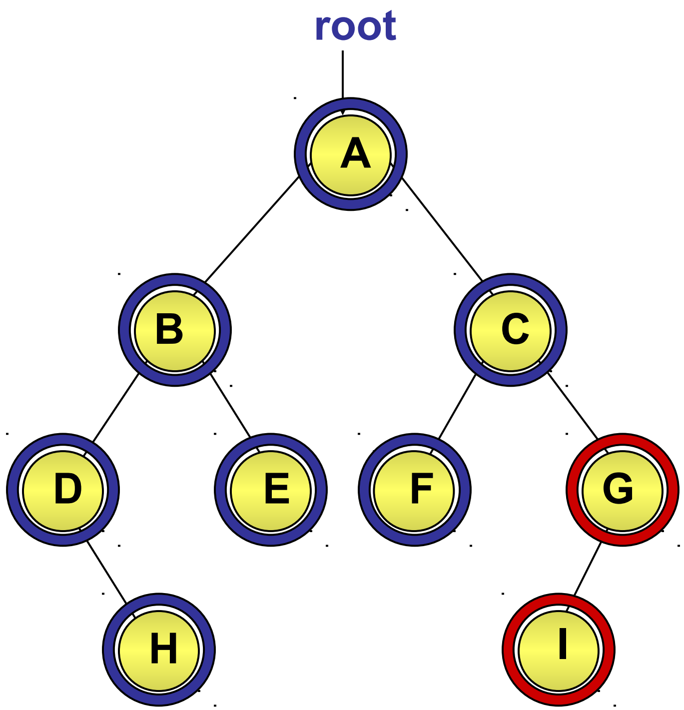
- Right subtree of C is not empty.
- Therefore, move to the right subtree of node C.



D H B E A F C

Inorder Traversal (Contd.)

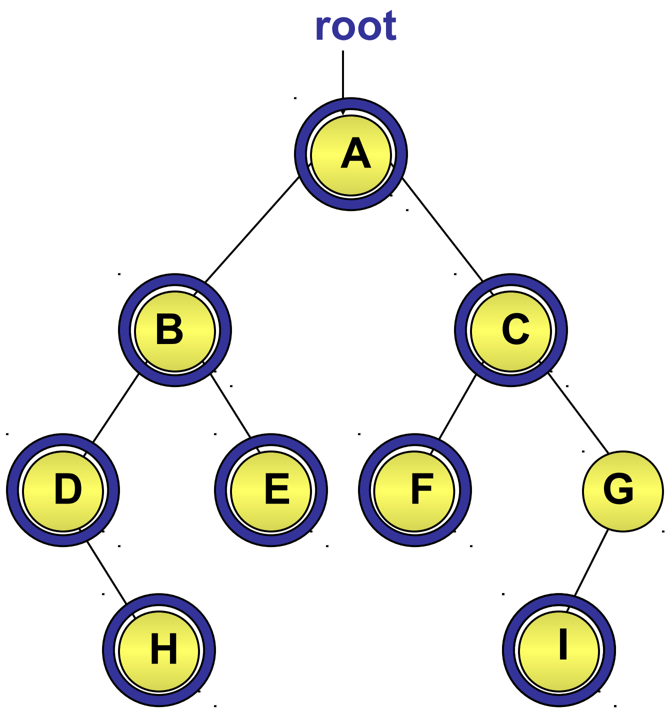
- Left subtree of G is not empty.
- Therefore, move to the left subtree of node G.



D H B E A F C

Inorder Traversal (Contd.)

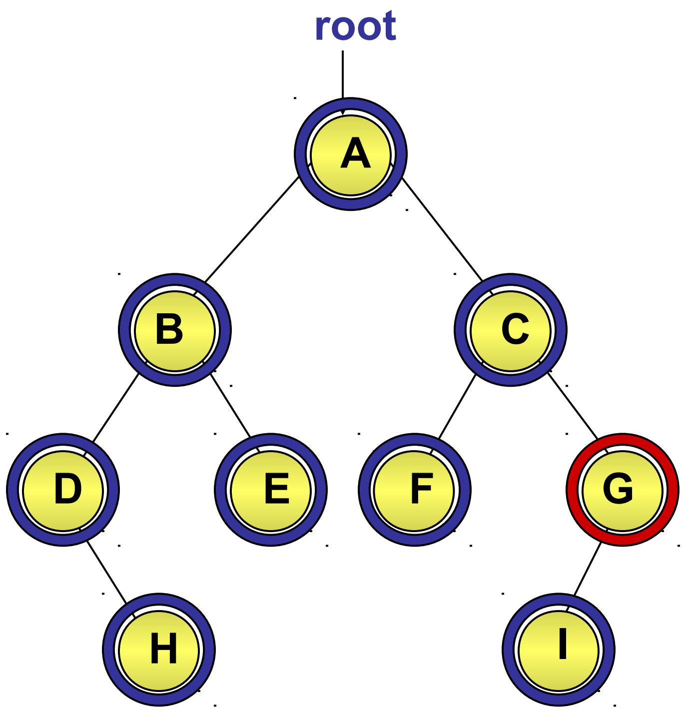
- Left subtree of I is empty.
- Therefore, visit I.



D H B E A F C I

Inorder Traversal (Contd.)

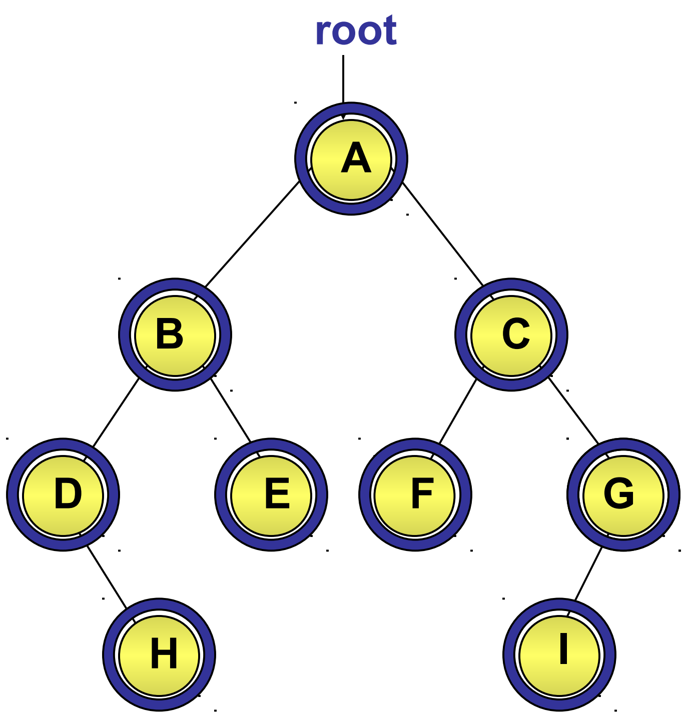
- Right subtree of I is empty.
- Therefore, move to node G.



D H B E A F C I

Inorder Traversal (Contd.)

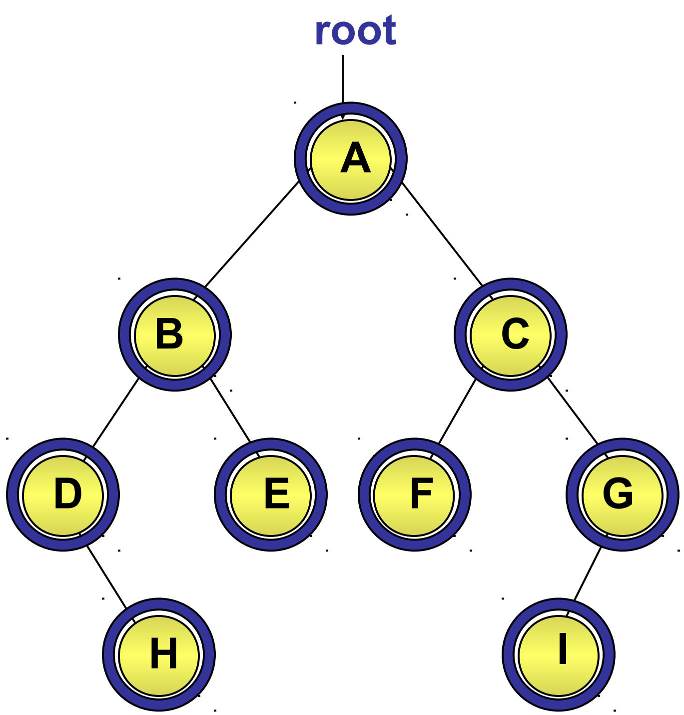
- Visit node G.



D H B E A F C I G

Inorder Traversal (Contd.)

- Right subtree of G is empty.



Traversal complete

D H B E A F C I G

ALGORITHM

ALGORITHM INORDERTRAVERSE()

{

1. set top=0, stack[top]=NULL, ptr = root
2. Repeat while ptr!=NULL
   - 2.1 set top=top+1
   - 2.2 set stack[top]=ptr
   - 2.3 set ptr=ptr->left
3. Set ptr=stack[top], top=top-1
4. Repeat while ptr!=NULL
   - 4.1 print ptr->info
   - 4.2 if ptr->right!=NULL then
     - 4.2.1set ptr=ptr->right
     - 4.2.2 goto step 2
   - 4.3 Set ptr=stack[top], top=top-1

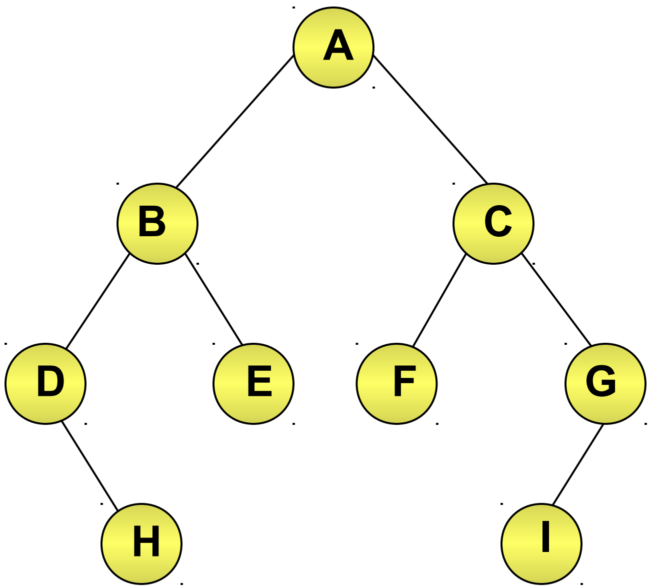
}

PREORDER TRAVERSAL

- Steps for traversing a tree in preorder sequence are as follows:
  1. Visit root
  2. Traverse the left subtree
  3. Traverse the right subtree

Preorder Traversal (Contd.)

- Perform the preorder traversal of the following tree.



Preorder Traversal: A B D H E C F G I

ALGORITHM

ALGORITHM PREORDERTRAVERSE()

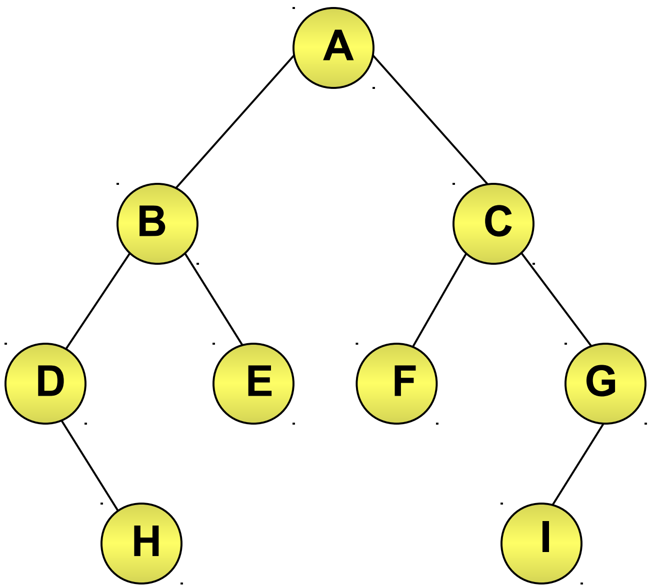
```
{
  1. set top=0, stack[top]=NULL, ptr = root
  2. Repeat while ptr!=NULL
     2.1 print ptr -> info
     2.2 if (ptr -> right != NULL)
            2.2.1 top = top +1
            2.2.2 set stack [ top] = ptr -> right
     2.3 if ( ptr -> left != NULL)
            2.3.1 ptr=ptr -> left
         else
            2.3.1 ptr=stack[top], top=top-1
}
```

Postorder Traversal

- Steps for traversing a tree in postorder sequence are as follows:
  1. Traverse the left subtree
  2. Traverse the right subtree
  3. Visit the root

Postorder Traversal (Contd.)

- Perform the postorder traversal of the following tree.



Postorder Traversal: H D E B F I G C A

ALGORITHM

ALGORITHM POSTORDERTRAVERSE()

{

1. set top = 0, stack [top] = NULL, ptr = root
2. Repeat while ptr!=NULL
   - 2.1 top = top +1 , stack [top] = ptr
   - 2.2 if (ptr -> right != NULL)
   - 2.2.1 top = top +1
   - 2.2.2 set stack [top] = - (ptr -> right)
   - 2.3 ptr = ptr -> left
3. ptr = stack [top], top = top-1
4. Repeat while (ptr > 0)
   - 4.1 print ptr -> info
   - 4.2 ptr = stack [top], top = top-1
5. if (ptr < 0)
   - 5.1 set ptr = - ptr
   - 5.2 Go to step 2

}

Recursive Traversal Implementation

```c
void print_preorder(tree t)
{
  if (NULL == t) {
  } else {
    printf("%c", t->value);
    print_preorder(t->left);
    print_preorder(t->right);
  }
}
```

```c
void print_preorder(tree t)
{
  if (NULL == t) {
  } else {
    printf("%c", t->value);
    print_preorder(t->left);
    print_preorder(t->right);
  }
}
```

```c
void print_inorder(tree t)
{
  if (NULL == t) {
  } else {
    print_inorder(t->left);
    printf("%c", t->value);
    print_inorder(t->right);
  }
}
```

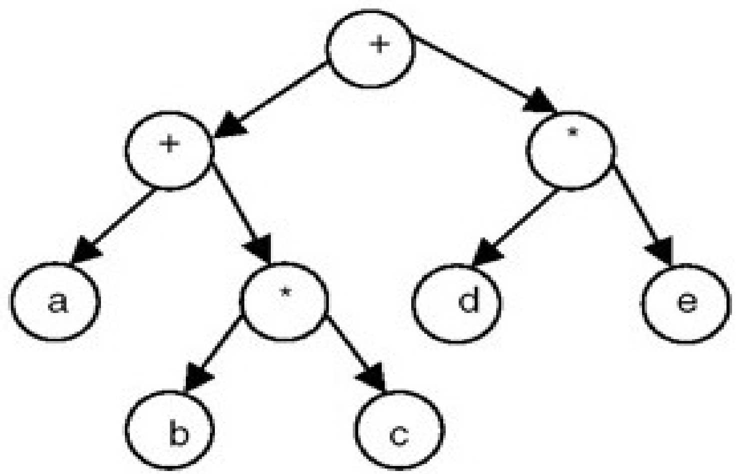
Just a minute

- In __________ traversal method, root is processed before traversing the left and right subtrees.

- Answer:
  - Preorder

Expression Binary Tree Traversal

If an expression is represented as a binary tree, the inorder traversal of the tree gives us an infix expression, whereas the postorder traversal gives us a postfix expression as shown in Figure.



Inorder : a + b * c + d * e

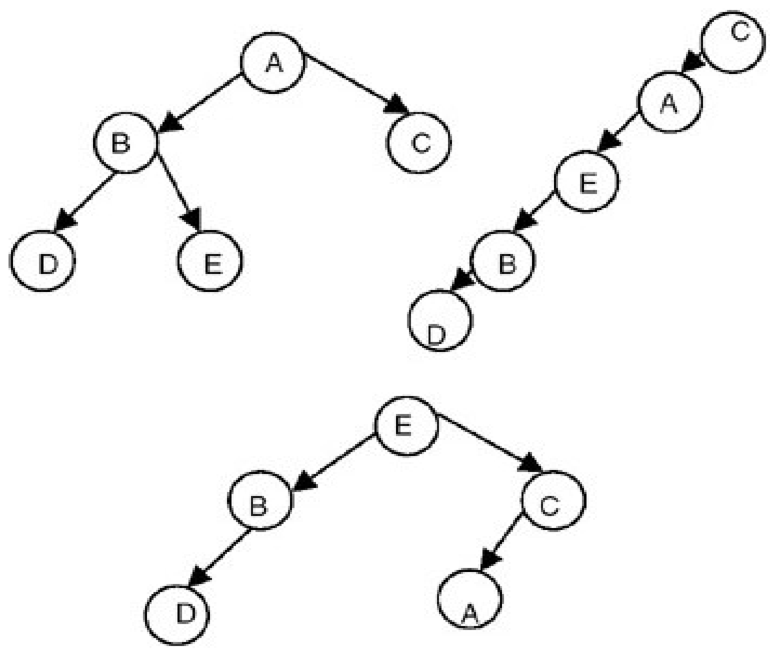
postorder : abc*+de*+

Construction of Binary Tree

Given an order of traversal of a tree, it is possible to construct a tree; for example, consider the folowing order:

Inorder = DBEAC

We can construct the binary trees shown in Figure by using this order of traversal



Construction of Binary Tree

Therefore, we conclude that given only one order of traversal of a tree, it is possible to construct a number of binary trees; a unique binary tree cannot be constructed with only one order of traversal.

For construction of a unique binary tree, we require two orders, in which one has to be inorder; the other can be preorder or postorder. For example, consider the following orders:

Inorder = DBEAC

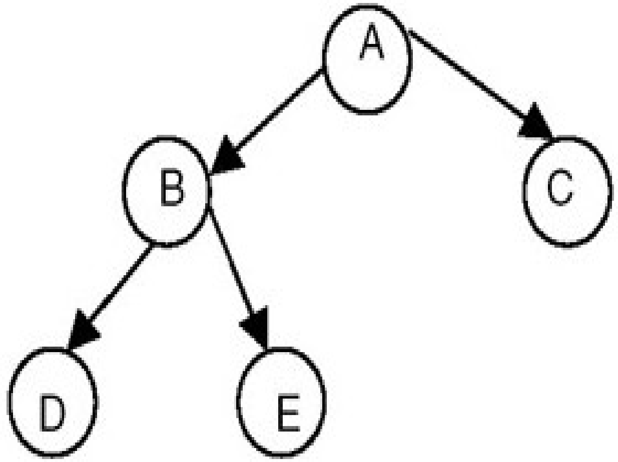
Postorder = DEBCA

Construction of Binary Tree

Inorder = DBEAC

Postorder = DEBCA

We can construct the unique binary tree shown in Figure by using these orders of traversal:

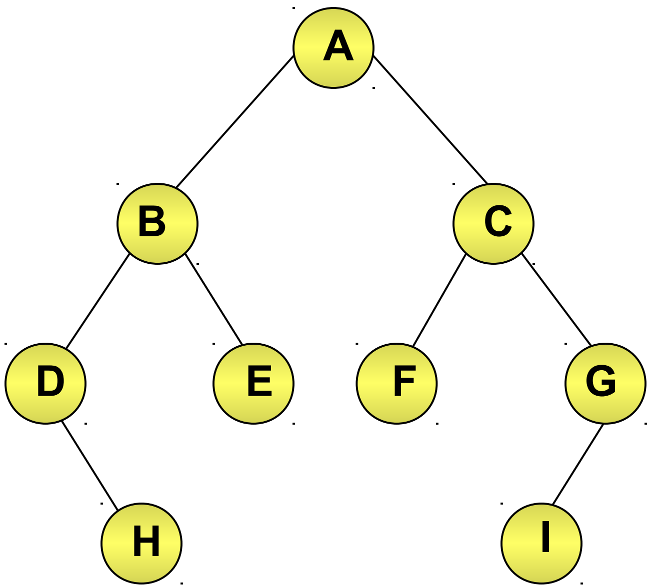


A unique binary tree constructed using its inorder and postorder.

Just a minute

- Construct the binary tree with the following:-
- Inorder:- D H B E A F C I G
- Postorder:- H D E B F I G C A

Postorder Traversal (Contd.)



Summary

- In this session, you learned that:
  - A tree is a nonlinear data structure that represents a hierarchical relationship among the various data elements.
  - A binary tree is a specific type of tree in which each node can have a maximum of two children.
  - Binary trees can be implemented by using arrays as well as linked lists, depending upon requirement.
  - Traversal of a tree is the process of visiting all the nodes of the tree once. There are three types of traversals, namely inorder, preorder, and postorder traversal.