INDIAN INSTITUTE OF INFORMATION TECHNOLOGY

Dharwad

ज्ञानेन विकासः

# Tree - Data Structures

Dr. Animesh Chaturvedi

Assistant Professor: IIIT Dharwad

Young Researcher: Heidelberg Laureate Forum

Postdoc: King's College London & The Alan Turing Institute

PhD: IIT Indore MTech: IIITDM Jabalpur

HEIDELBERG LAUREATE FORUM

KING'S College LONDON

Indian Institute of Technology Indore

भारतीय प्रौद्योगिकी संस्थान इंदौर

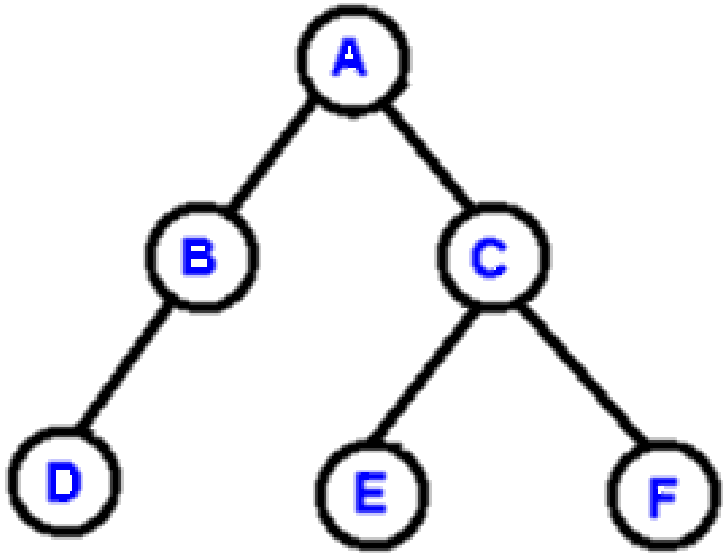
The Alan Turing Institute

PDPM

Indian Institute of Information Technology, Design and Manufacturing, Jabalpur

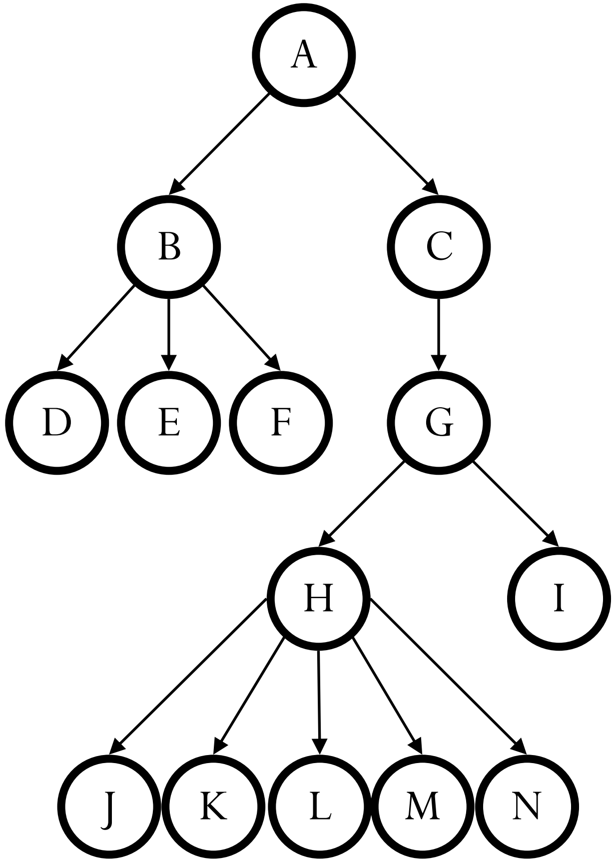
# Trees

- Tree is special kind of Graph.
- Family Trees
- Organization Charts
- Classification trees
  - what kind of flower is this?
  - is this mushroom poisonous?
- File directory structure
  - folders, subfolders in Windows
  - directories, subdirectories in UNIX
- Non-recursive procedure call chains



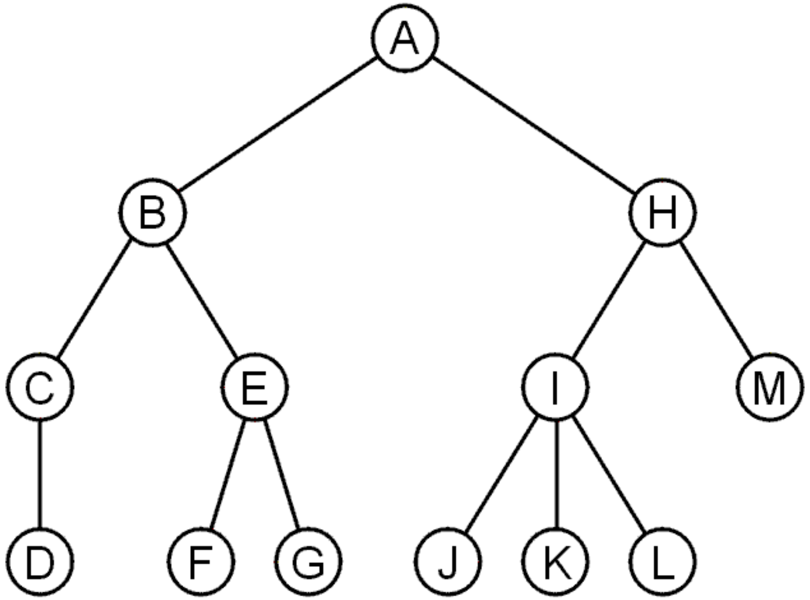
# Tree Terminology

*root:*
*leaf:*
*child:*
*parent:*
*sibling:*
*ancestor:*
*descendent:*
*subtree:*
*depth:*
*height:*
*degree:*
*branching factor:*
*preorder traversal:*
*postorder traversal:*



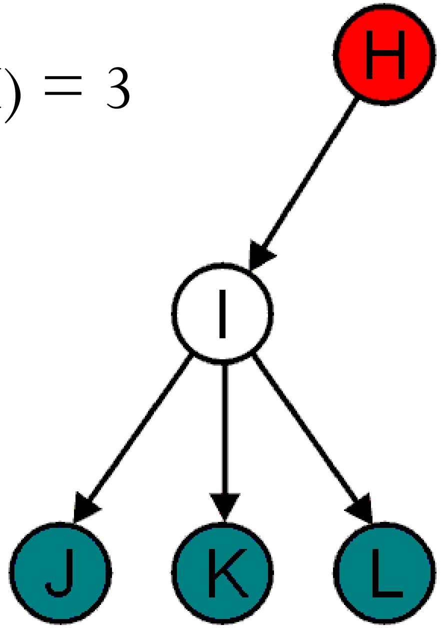
# Trees

- A rooted tree data structure stores information in *nodes*
  - Similar to linked lists:
    - There is a first node, or *root*
    - Each node has variable number of references to successors
    - Each node, other than the root, has exactly one node pointing to it



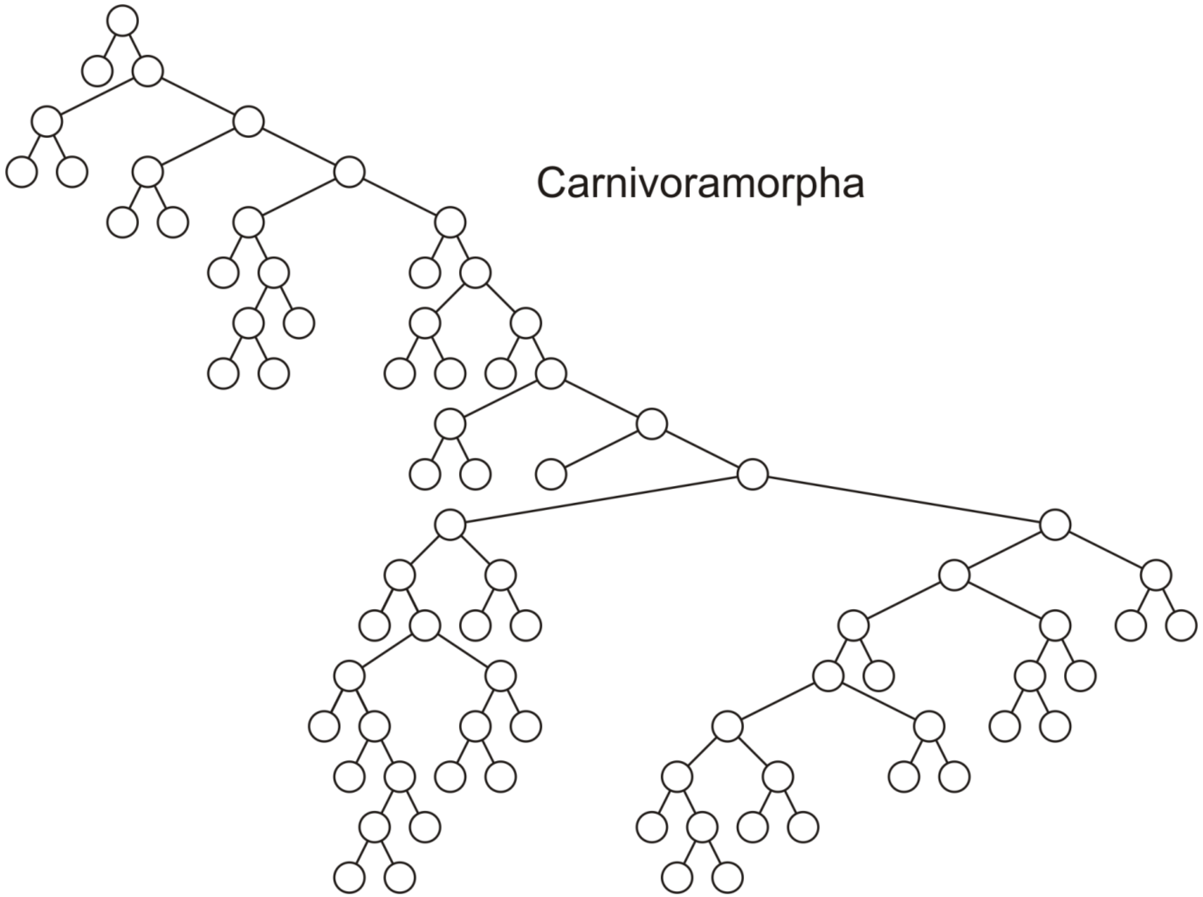
# Terminology

- Each node can be connected to arbitrary number of nodes, called ***children***
- All nodes will have zero or more child nodes or *children*
  - I has three children: J, K and L
- For all nodes other than the root node, there is one parent node
  - H is the parent I
- The *degree* of a node is defined as the number of its children: deg(I) = 3
- Nodes with the same parent are *siblings*
  - J, K, and L are siblings



# Terminology

- Phylogenetic trees have nodes with degree 2 or 0:

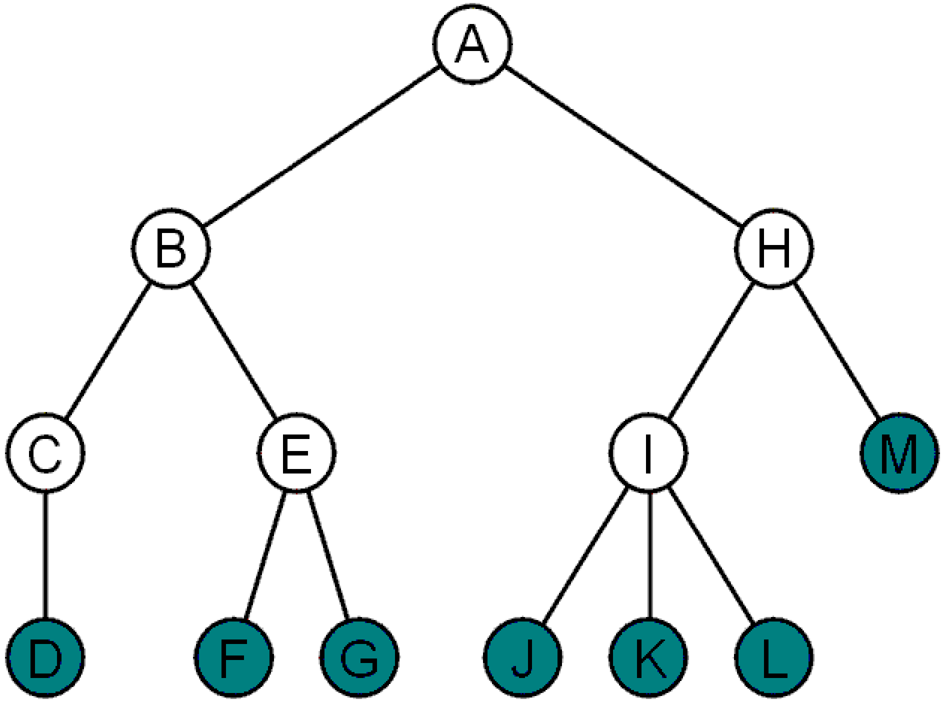



Wesley-Hunt, G. D.; Flynn, J. J. “Phylogeny of the Carnivora: basal relationships among the Carnivoramorphans, and assessment of the position of ‘Miacoidea’

# Terminology

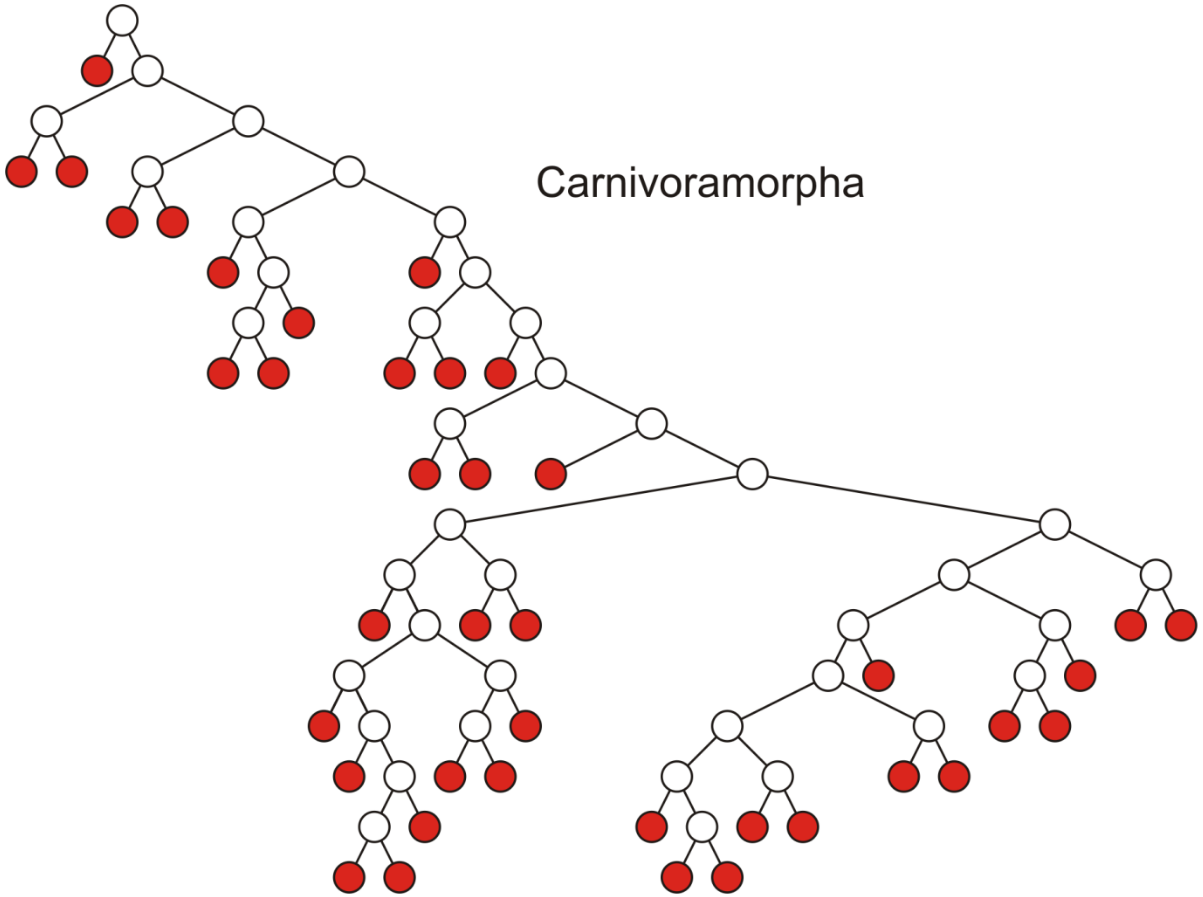
- Nodes with degree zero are also called *leaf nodes*

- All other nodes are said to be *internal nodes*, that is, they are internal to the tree



# Terminology

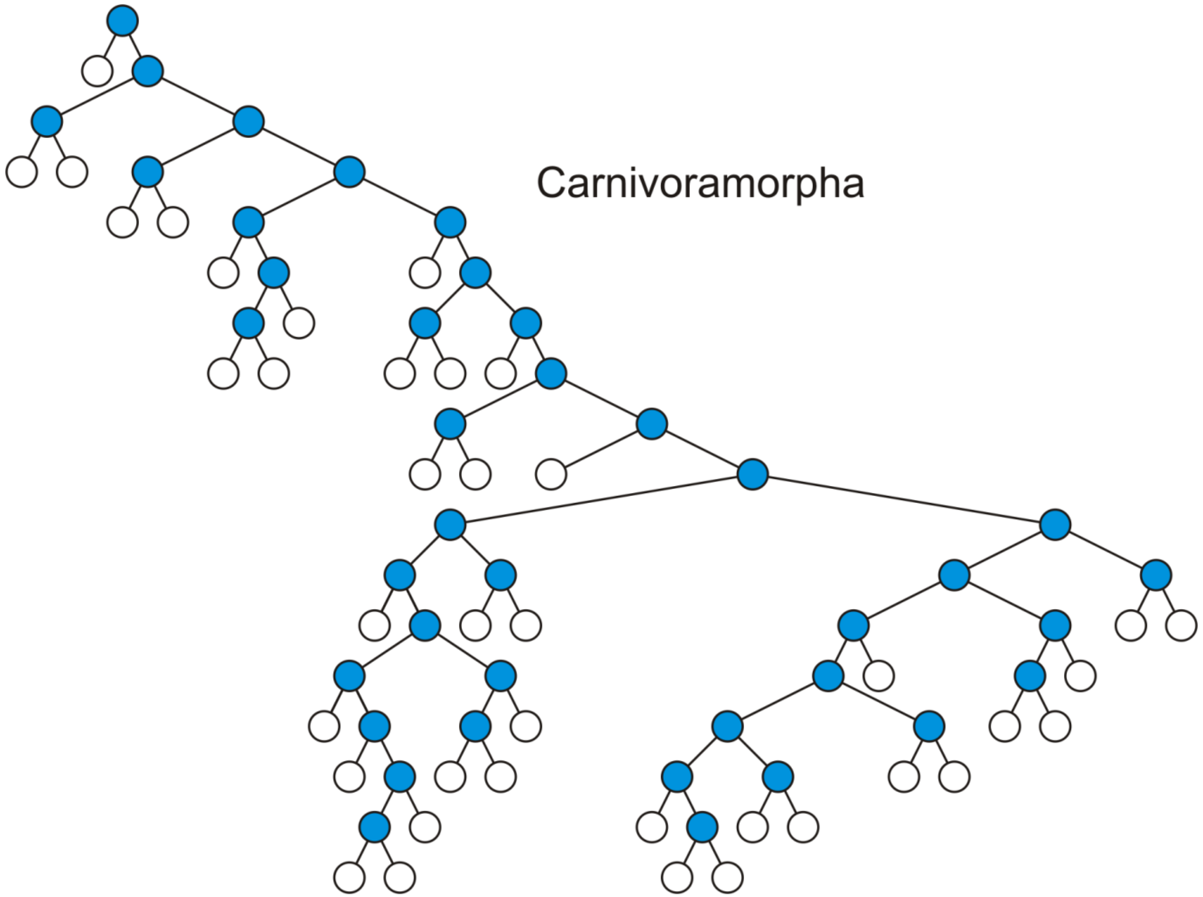
Leaf nodes:




Wesley-Hunt, G. D.; Flynn, J. J. "Phylogeny of the Carnivora: basal relationships among the Carnivoramorphans, and assessment of the position of 'Miacoidea'

# Terminology

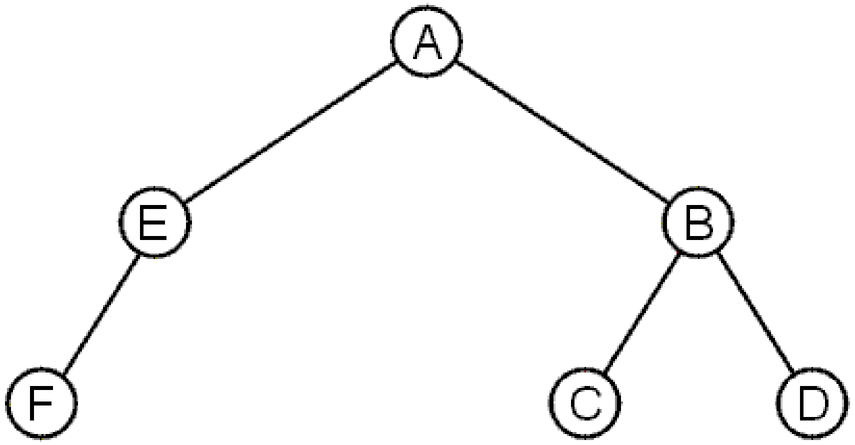
Internal nodes:

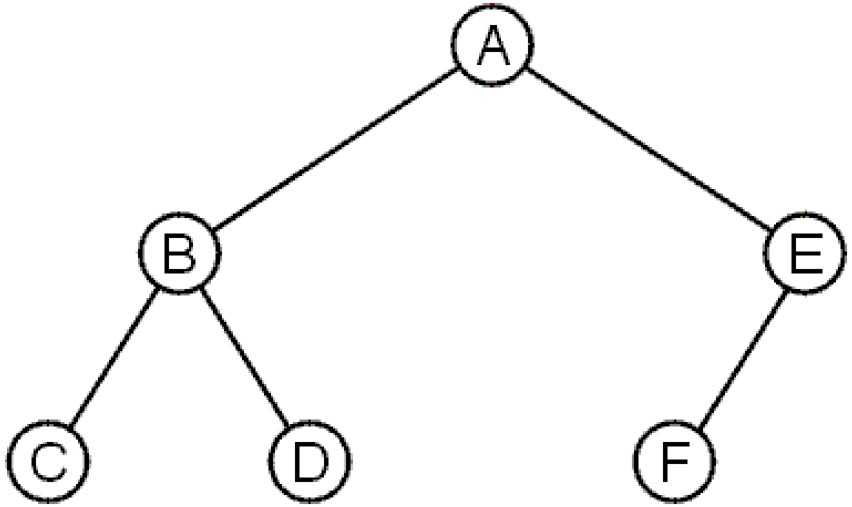



Wesley-Hunt, G. D.; Flynn, J. J. “Phylogeny of the Carnivora: basal relationships among the Carnivoramorphans, and assessment of the position of ‘Miacoidea’

# Terminology

- These trees are equal if the order of the children is ignored
  - *unordered trees*

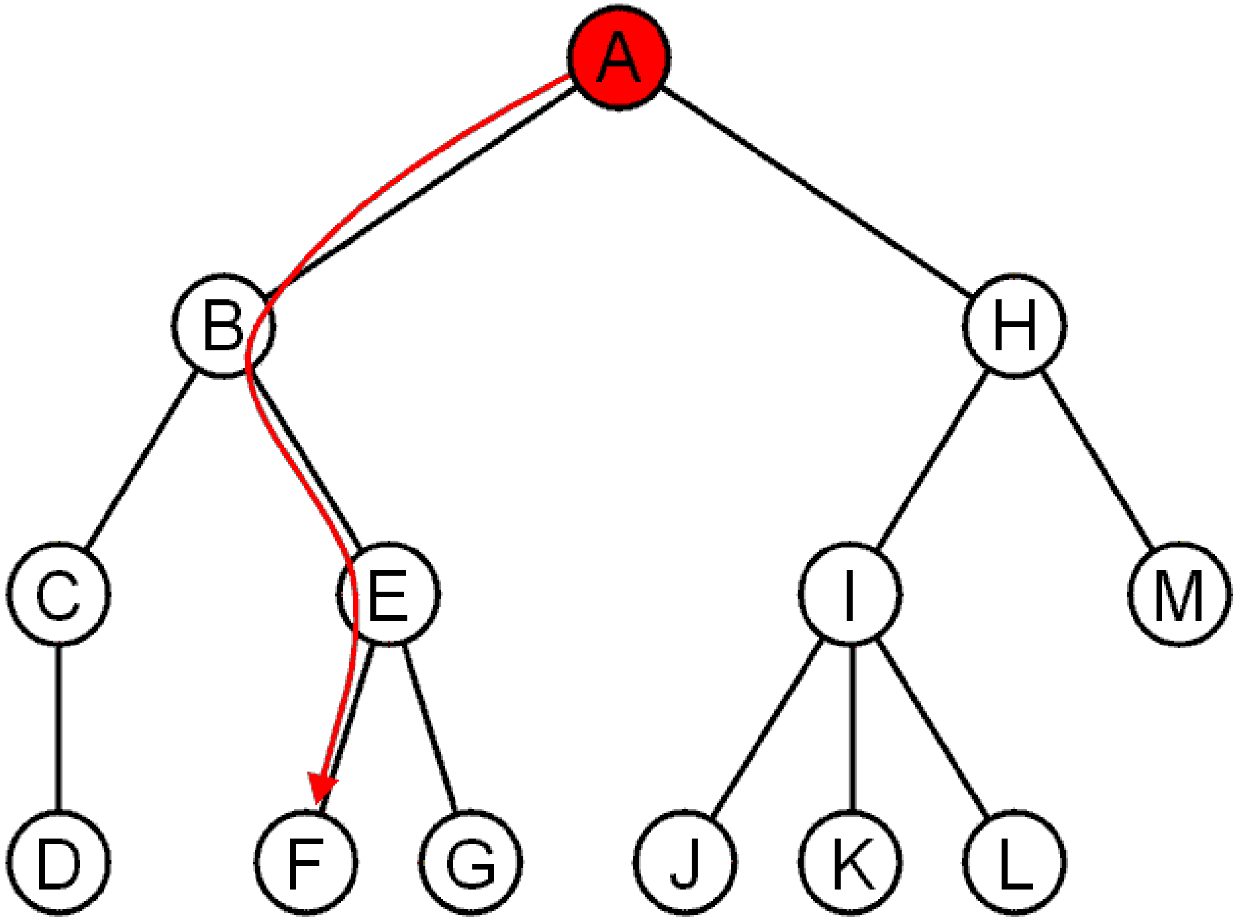



- They are different if order is relevant (*ordered trees*)
  - We will usually examine ordered trees (linear orders)
  - In a hierarchical ordering, order is not relevant

# Terminology

- The shape of a rooted tree gives a natural flow from the *root node*, or just *root*



# Terminology

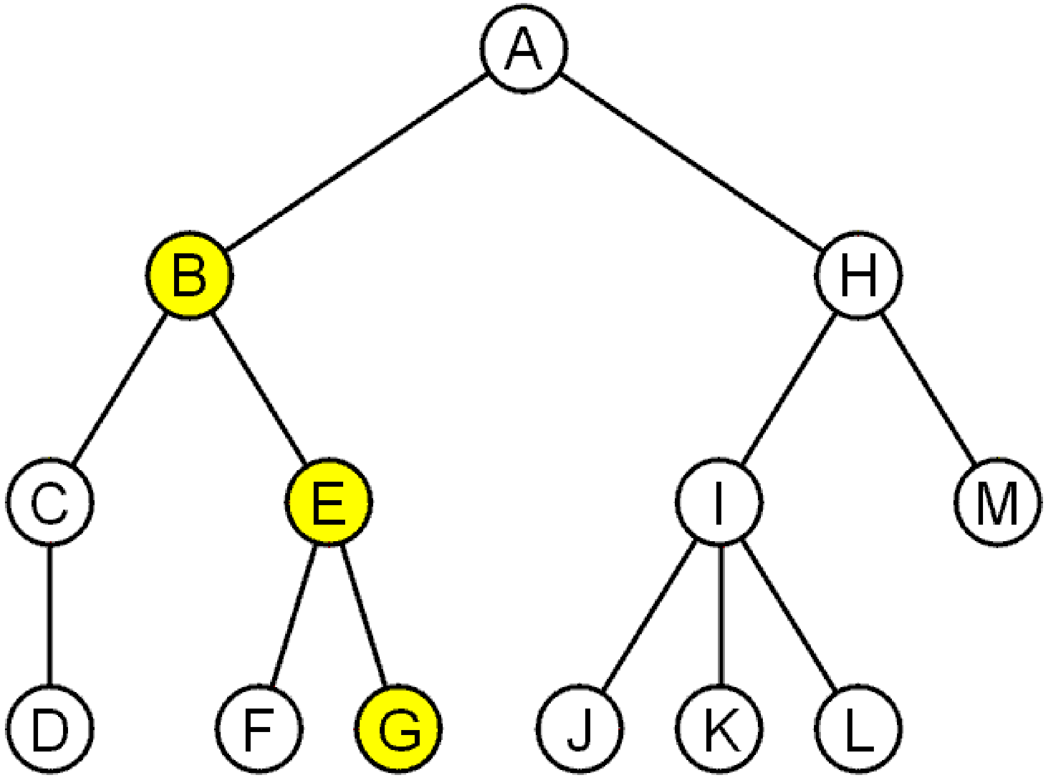
- A path is a sequence of nodes

$$(a_0, a_1, \ldots, a_n)$$

where $a_{k+1}$ is a child of $a_k$ is

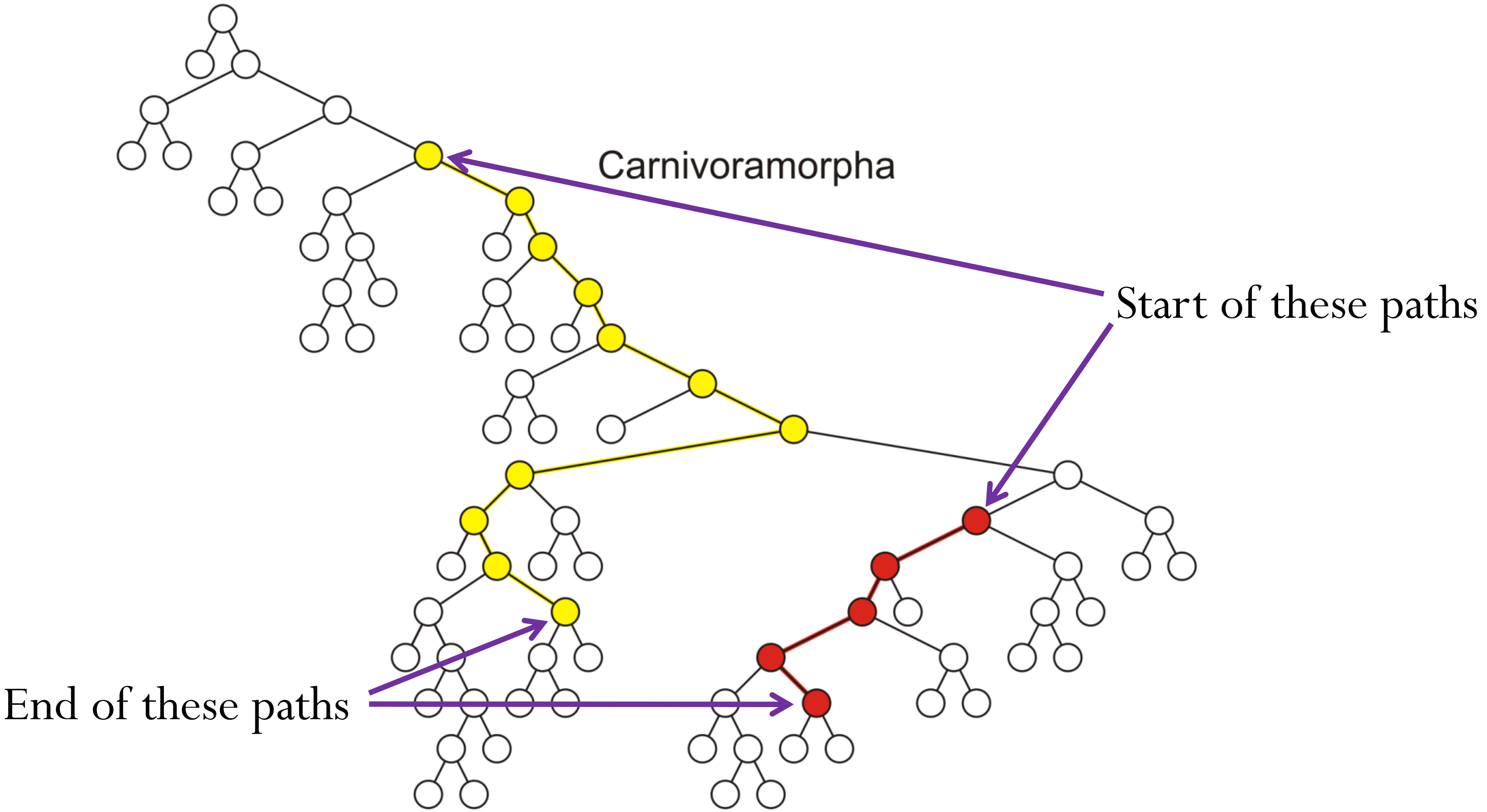
- The length of this path is $n$

- *E.g.*, the path (B, E, G) has length 2



# Terminology

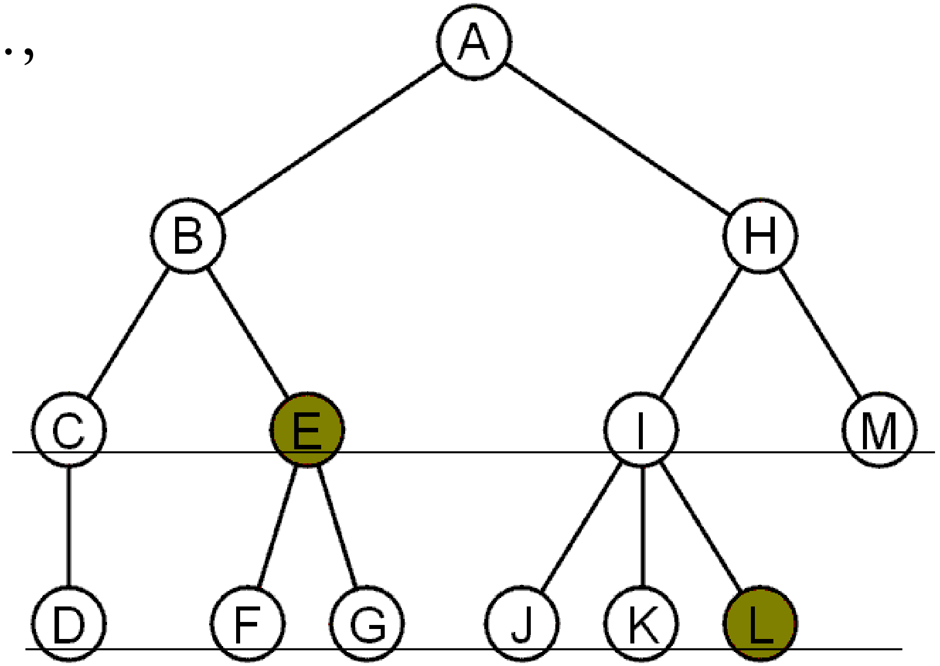
- Paths of length 10 (11 nodes) and 4 (5 nodes)




Wesley-Hunt, G. D.; Flynn, J. J. "Phylogeny of the Carnivora: basal relationships among the Carnivoramorphans, and assessment of the position of 'Miacoidea'

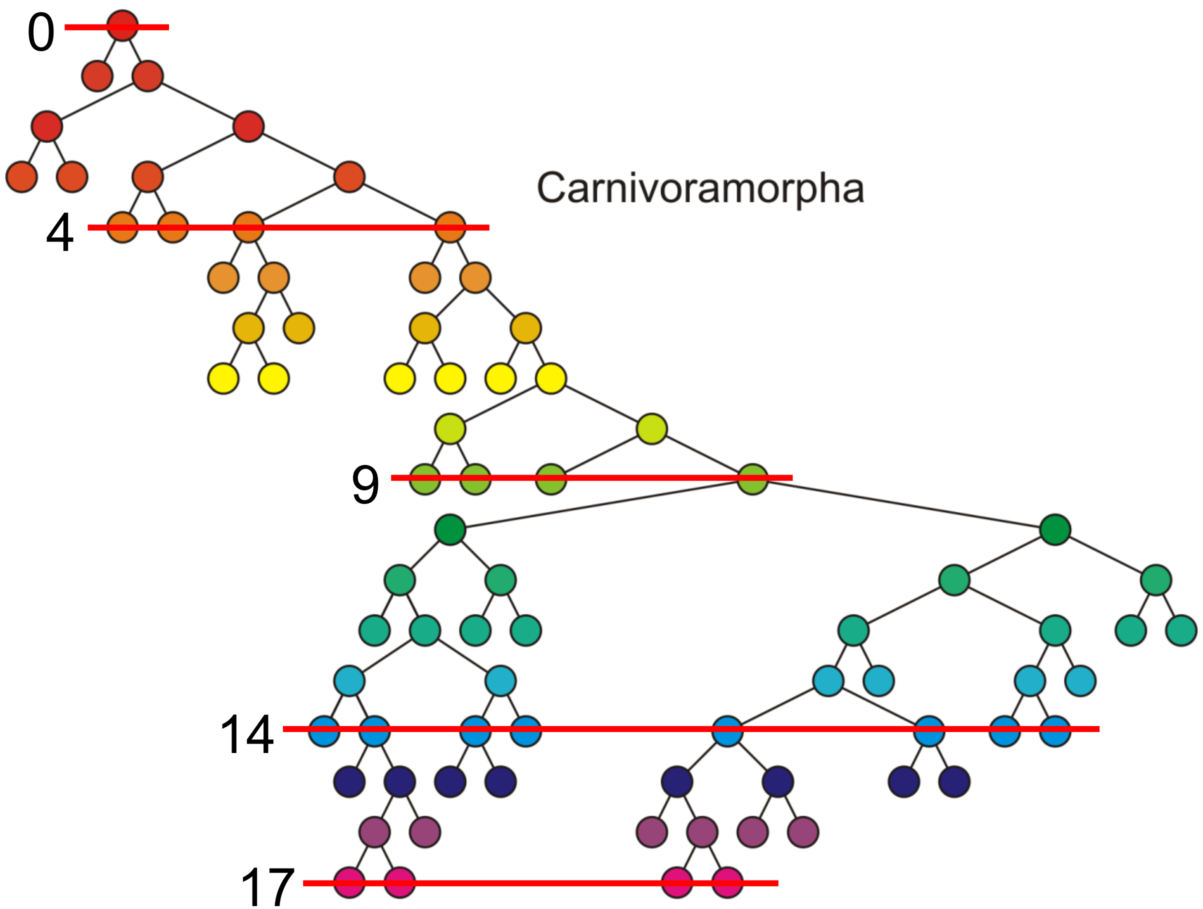
# Tree Terminology

- The depth of a node is the number of edges from the root to the node.
- The height of a node is the number of edges from the node to the deepest leaf.
- The height of a tree is a height of the root.
- For each node in a tree, there exists a unique path from the root node to that node
- The length of this path is the *depth* of the node, *e.g.*,
  - E has depth 2
  - L has depth 3



# Terminology

- Nodes of depth up to 17



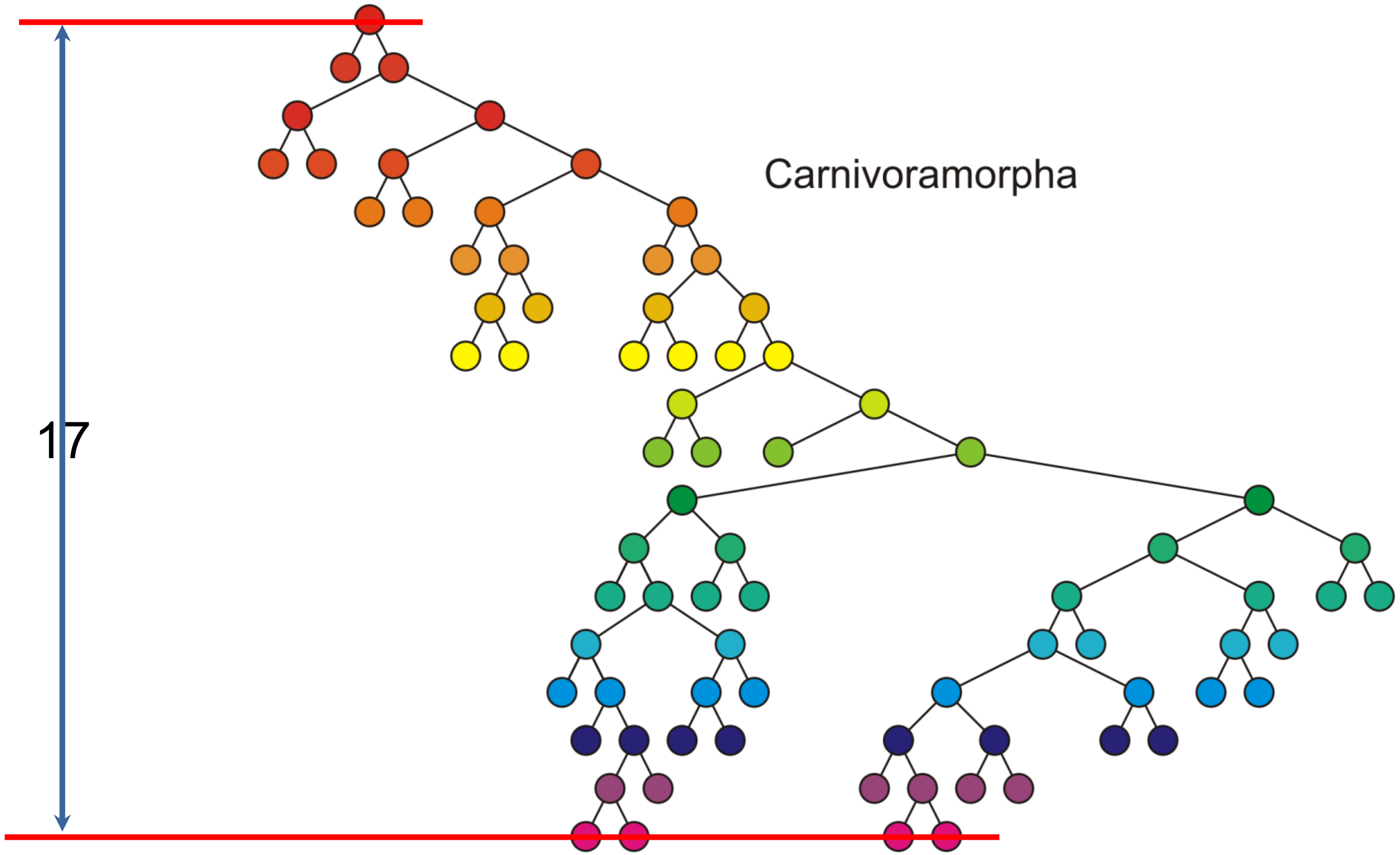

Wesley-Hunt, G. D.; Flynn, J. J. “Phylogeny of the Carnivora: basal relationships among the Carnivoramorphans, and assessment of the position of ‘Miacoidea’

# Terminology

- The *height* of a tree is defined as the maximum depth of any node within the tree
- The height of a tree with one node is 0
  - Just the root node
- For convenience, we define the height of the empty tree to be $-1$

# Terminology

- The *height* of this tree is 17



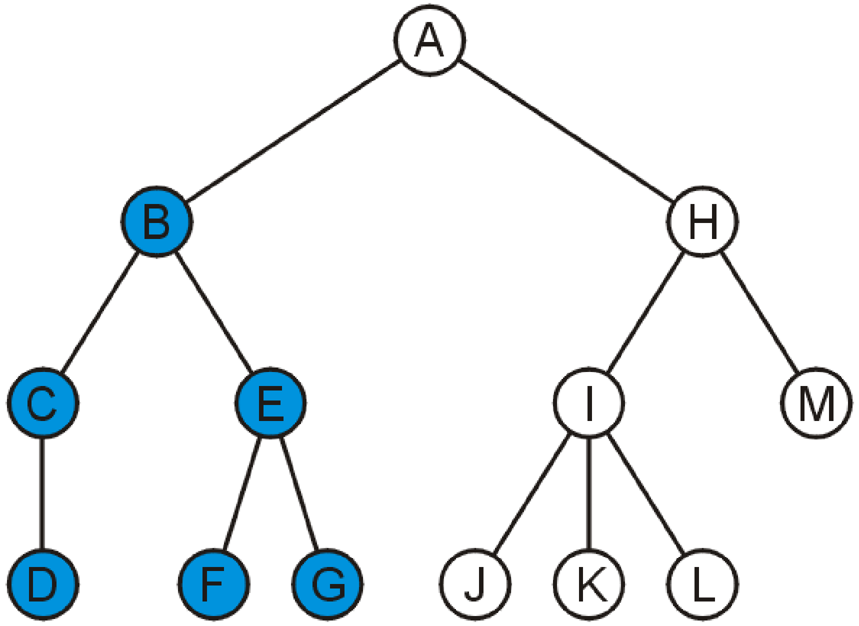

Wesley-Hunt, G. D.; Flynn, J. J. “Phylogeny of the Carnivora: basal relationships among the Carnivoramorphans, and assessment of the position of ‘Miacoidea’

# Terminology

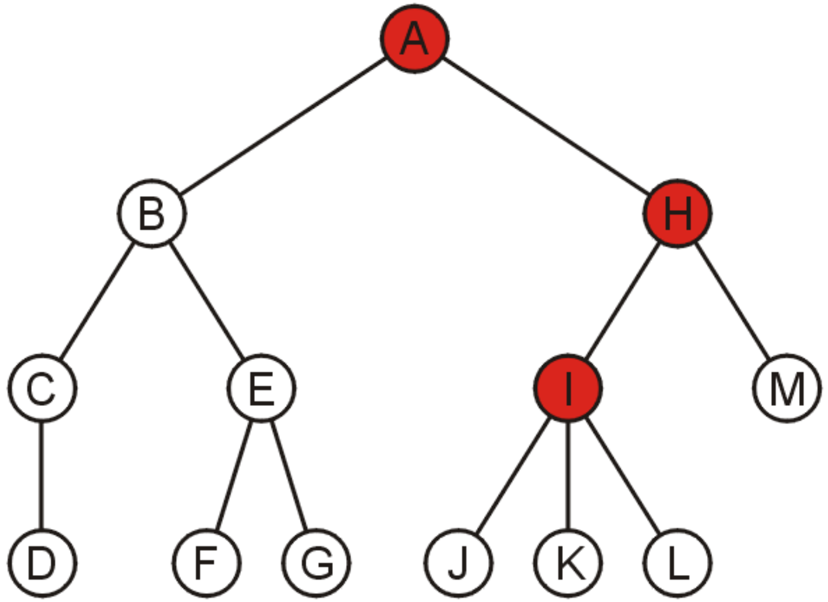
- If a path exists from node $a$ to node $b$:
  - $a$ is an *ancestor* of $b$
  - $b$ is a *descendent* of $a$
- Thus, a node is both an ancestor and a descendant of itself
  - We can add the adjective *strict* to exclude equality: $a$ is a *strict descendent* of $b$ if $a$ is a descendant of $b$ but $a \neq b$
- The root node is an ancestor of all nodes

# Terminology

- The descendants of node B are B, C, D, E, F, and G:

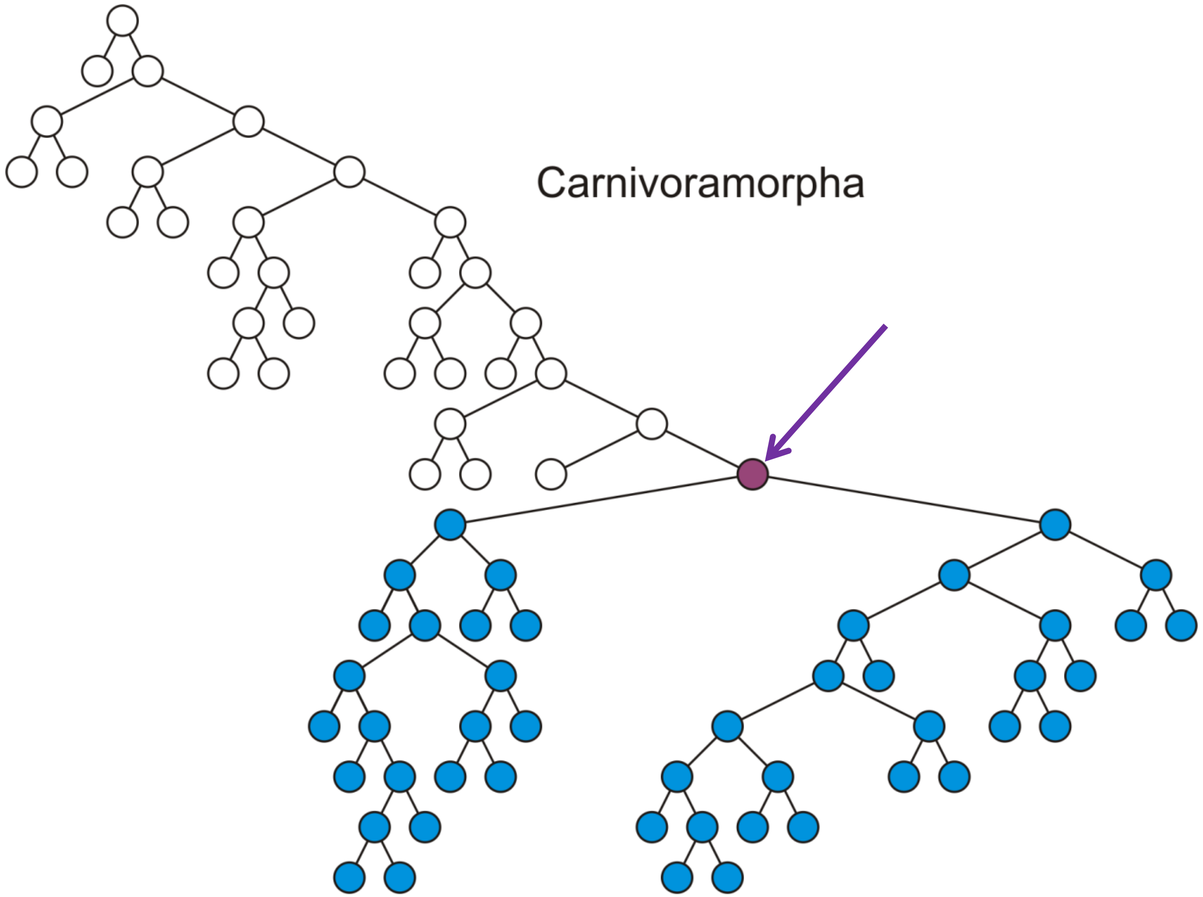



- The ancestors of node I are I, H, and A:



# Terminology

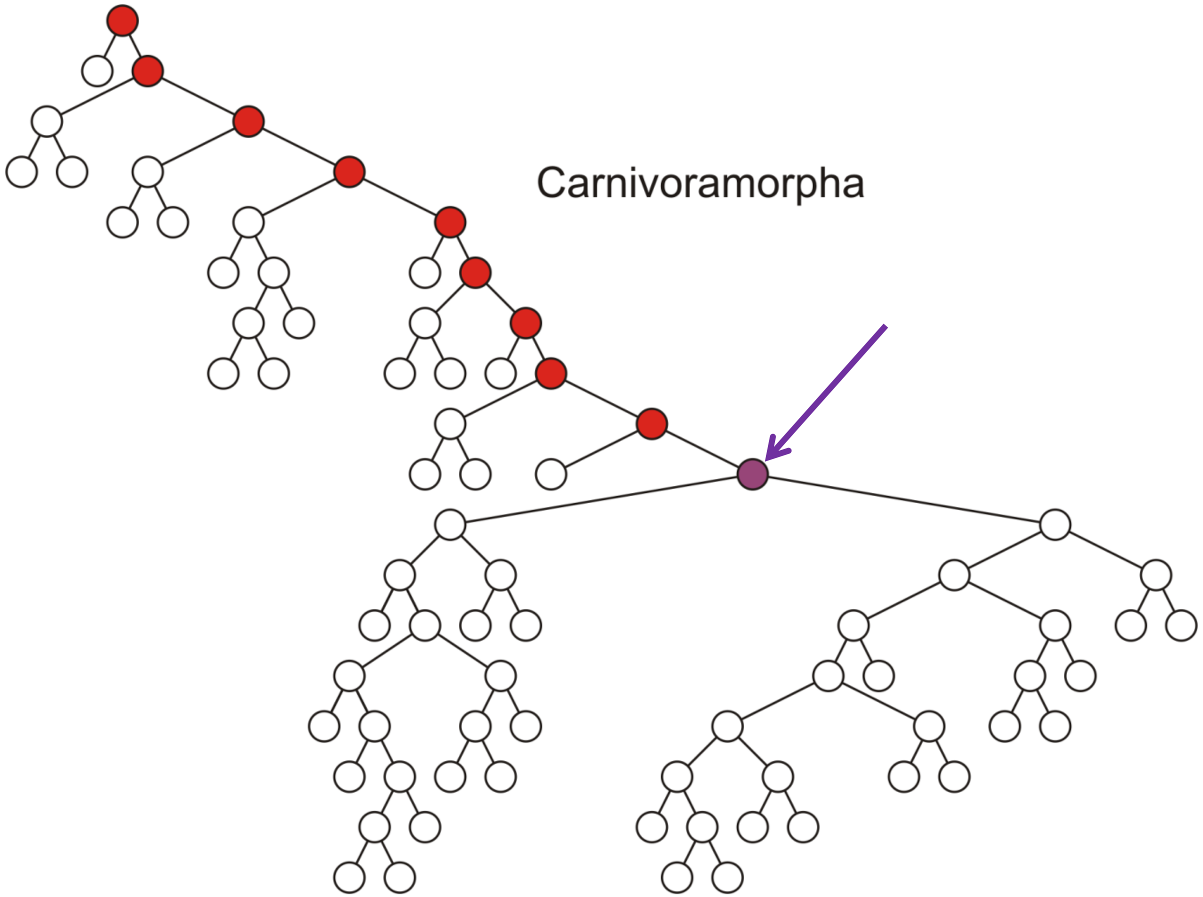
All descendants (including itself) of the indicated node




Wesley-Hunt, G. D.; Flynn, J. J. "Phylogeny of the Carnivora: basal relationships among the Carnivoramorphans, and assessment of the position of 'Miacoidea'

# Terminology

All ancestors (including itself) of the indicated node

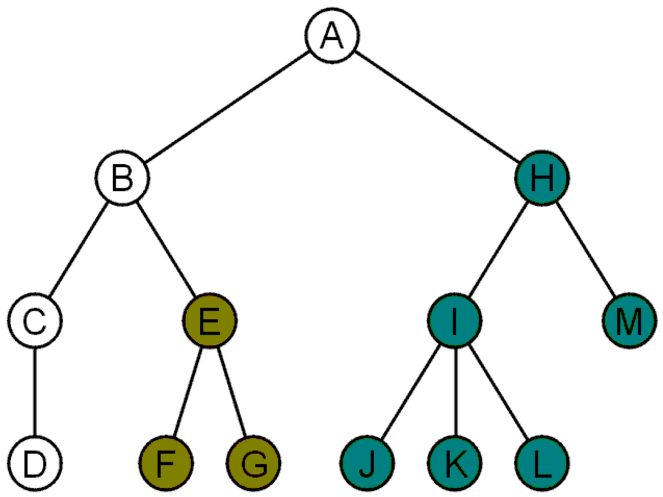



Wesley-Hunt, G. D.; Flynn, J. J. "Phylogeny of the Carnivora: basal relationships among the Carnivoramorphans, and assessment of the position of 'Miacoidea'

# Terminology

- Another approach to a tree is to define the tree recursively:
  - A degree-0 node is a tree
  - A node with degree $n$ is a tree if it has $n$ children and all of its children are disjoint trees (*i.e.*, with no intersecting nodes)

- Given any node $a$ within a tree with root $r$, the collection of $a$ and all of its descendants is said to be a *subtree of the tree with root a*



# Example: XHTML and CSS

- The XML of XHTML has a tree structure
- Cascading Style Sheets (CSS) use the tree structure to modify the display of HTML

# Example: XHTML and CSS

- Consider the following XHTML document

```
<html>
    <head>
        <title>Hello World!</title>
    </head>
    <body>
        <h1>This is a <u>Heading</u></h1>

        <p>This is a paragraph with some
        <u>underlined</u> text.</p>
    </body>
</html>
```

# Example: XHTML and CSS

- Consider the following XHTML document

title

```
<html>
    <head>
        <title>Hello World!</title>
    </head>
    <body>
        <h1>This is a <u>Heading</u></h1>

        <p>This is a paragraph with some
        <u>underlined</u> text.</p>
    </body>
</html>
```

heading

body of page

underlining

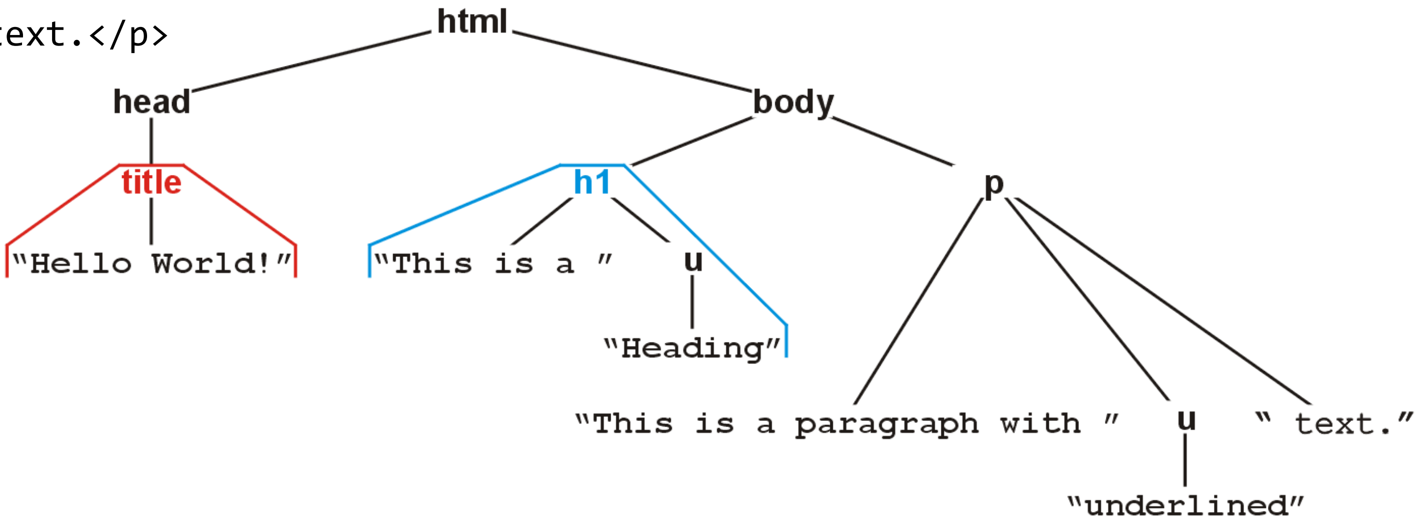
paragraph

# Example: XHTML and CSS

- The nested tags define a tree rooted at the HTML tag

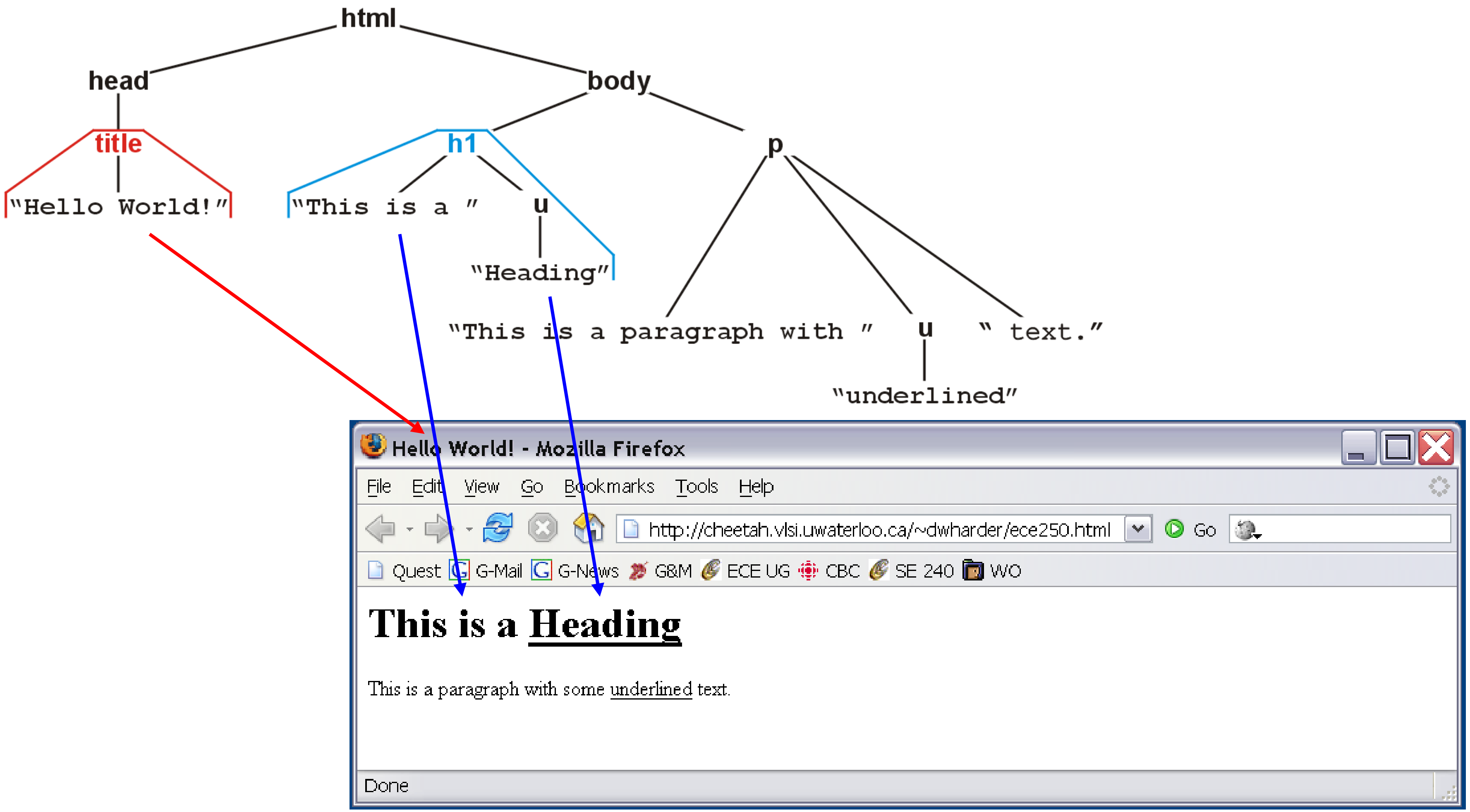
```
<html>
    <head>
        <title>Hello World!</title>
    </head>
    <body>
        <h1>This is a <u>Heading</u></h1>

        <p>This is a paragraph with some
        <u>underlined</u> text.</p>
    </body>
</html>
```



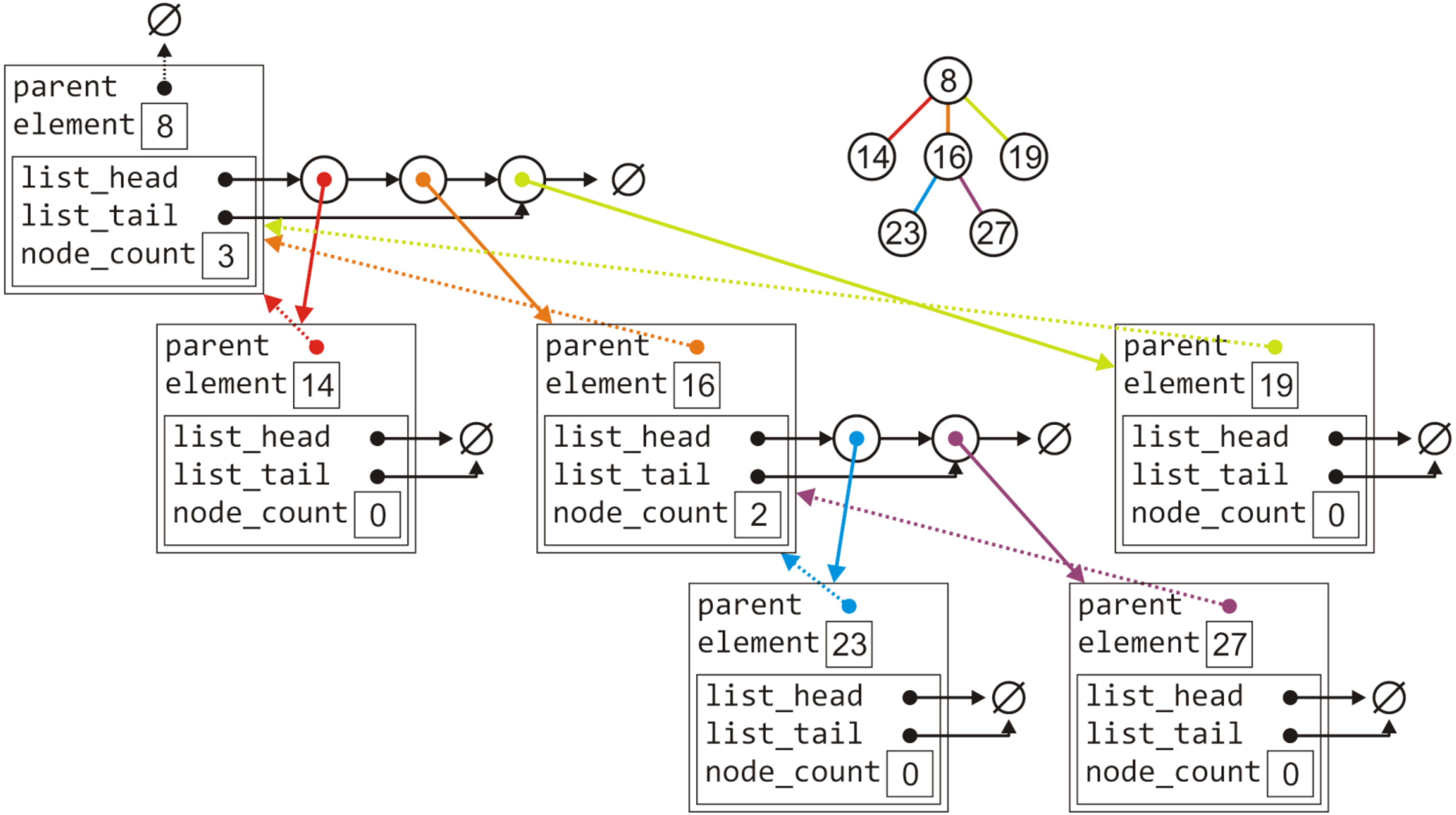
# Example: XHTML and CSS

- Web browsers render  this tree as a web page



# Tree Structure

- The tree with six nodes would be stored as follows:



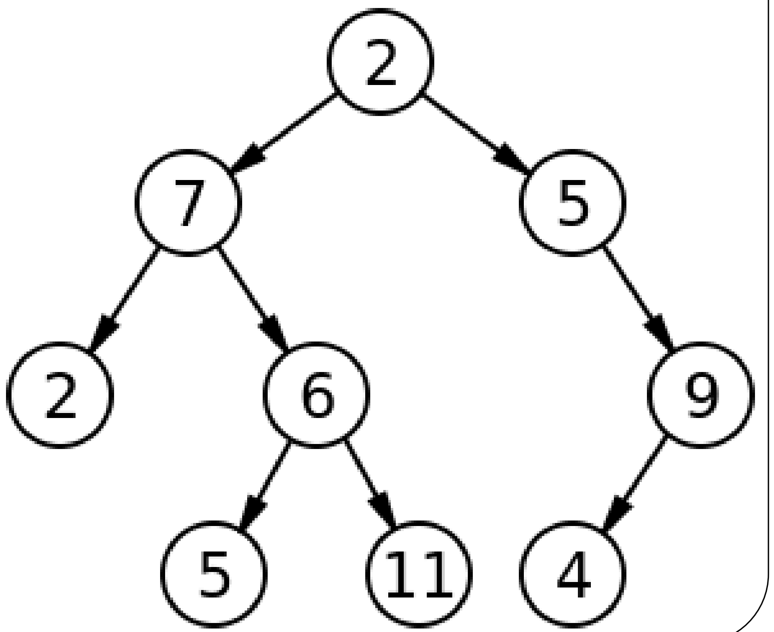
# Advantages of Trees

- Trees are so useful and frequently used, because they have some very serious advantages:-
  - Trees reflect structural relationships in the data
  - Trees are used to represent hierarchies
  - Trees provide an efficient insertion and searching
  - Trees are very flexible data, allowing to move subtrees around with minimum effort

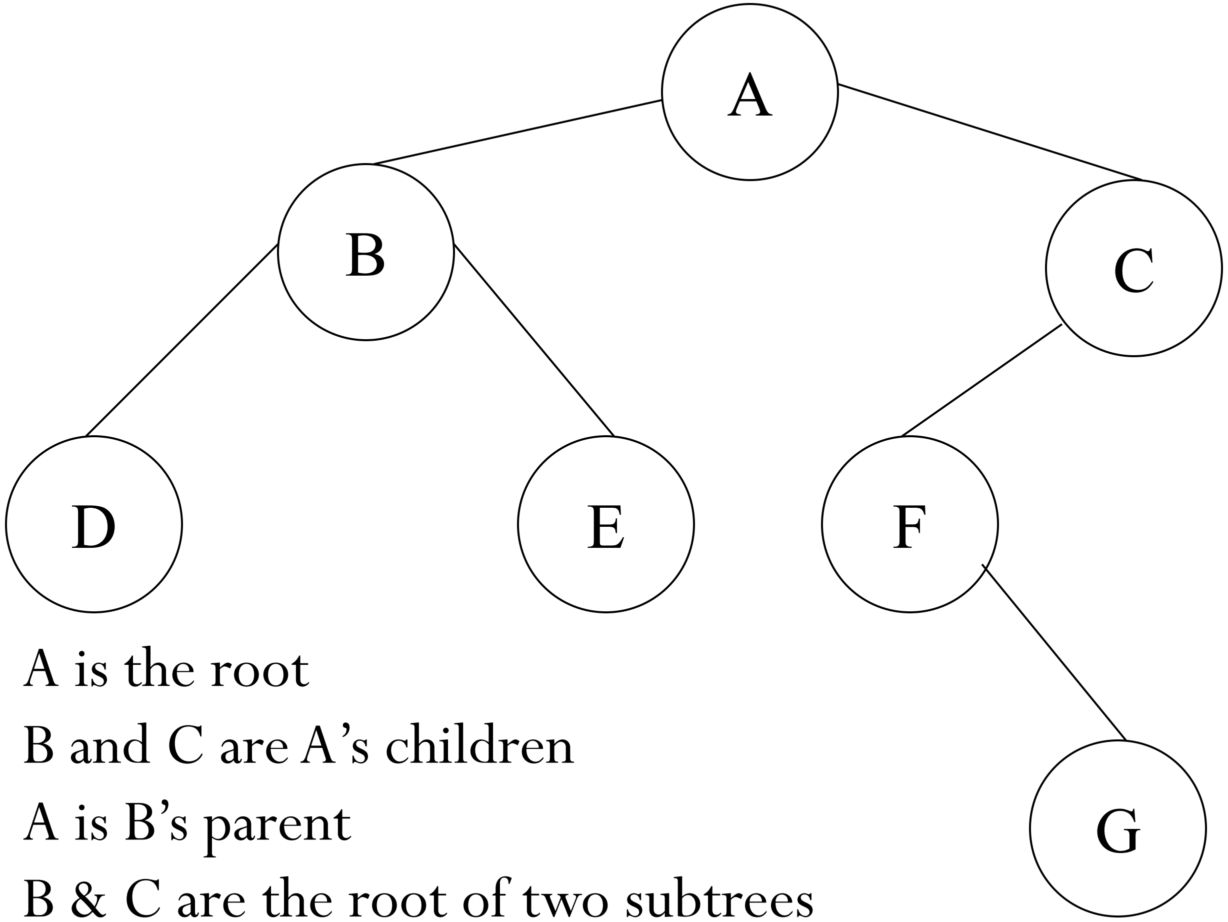
# Binary Tree

# Definitions and Terminologies

- A data structure in which a record is linked to two successor records, usually referred to as the left branch when greater and the right when less than the previous record.
- A binary tree is made of nodes, where each node contains a "left" reference, a "right" reference, and a data element. The topmost node in the tree is called the ***root***
- Every node (excluding a root) in a tree is connected by a directed edge from exactly one other node which is called a ***parent***
- Each node can be connected to two nodes, called ***children***
- Nodes with no children are called ***leaves*** or ***external nodes***
- Nodes which are not leaves are called ***internal nodes***
- Nodes with the same parent are called ***sibling***



# Definitions and Terminologies




A is the root

B and C are A's children

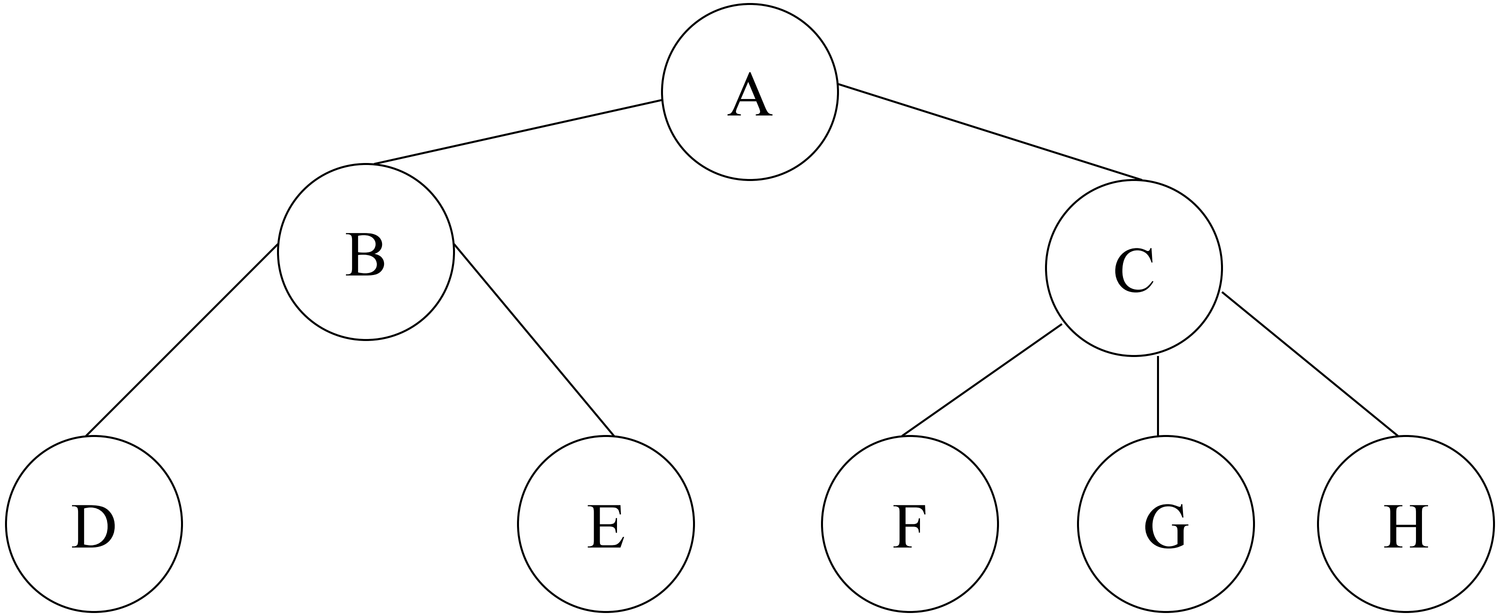
A is B's parent

B & C are the root of two subtrees

D, E, and G are leaves

# Definitions and Terminologies

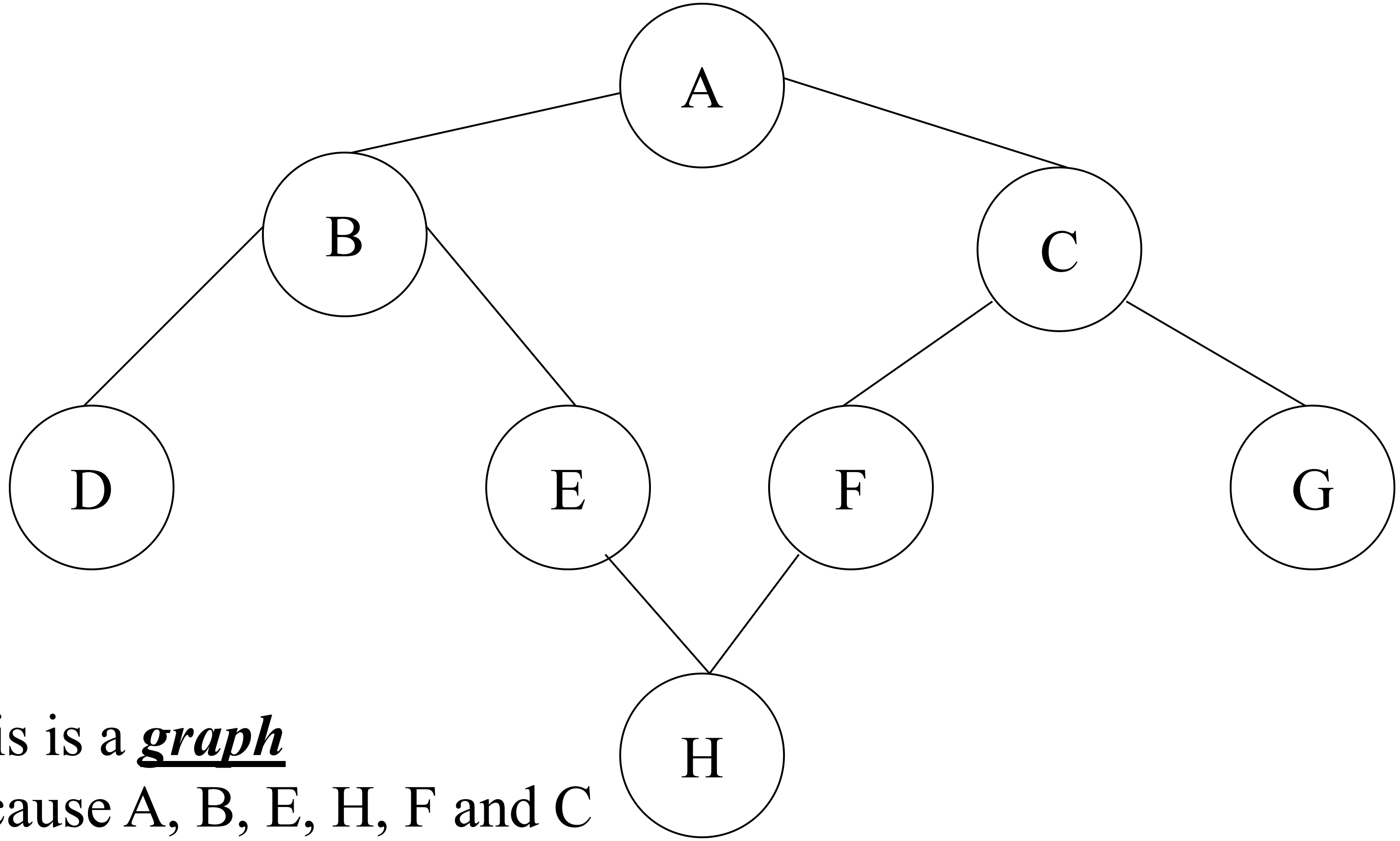
This is ***NOT*** A Binary Tree




This is a ***general tree*** because C has three child nodes

# Definitions and Terminologies

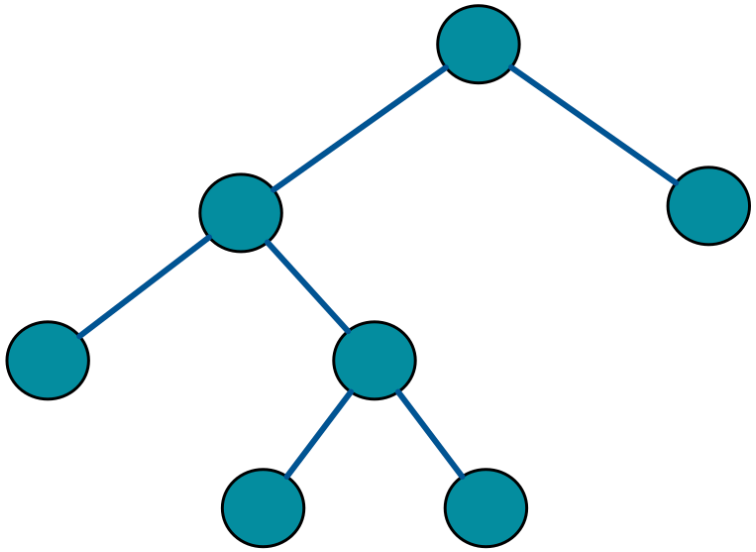
This is ***NOT*** A Binary Tree

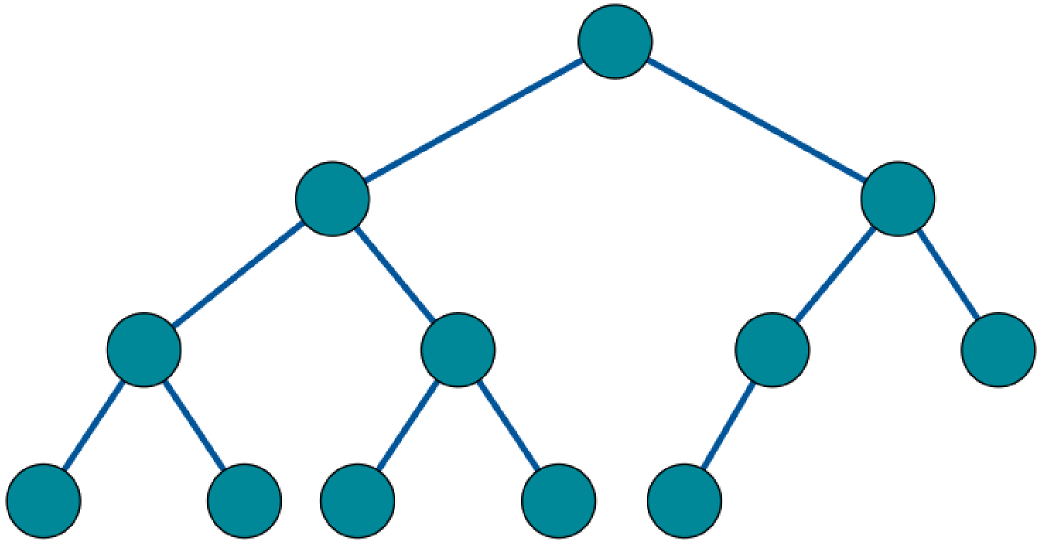



This is a ***graph***
because A, B, E, H, F and C
form a circuit

# Full & Complete Binary Tree

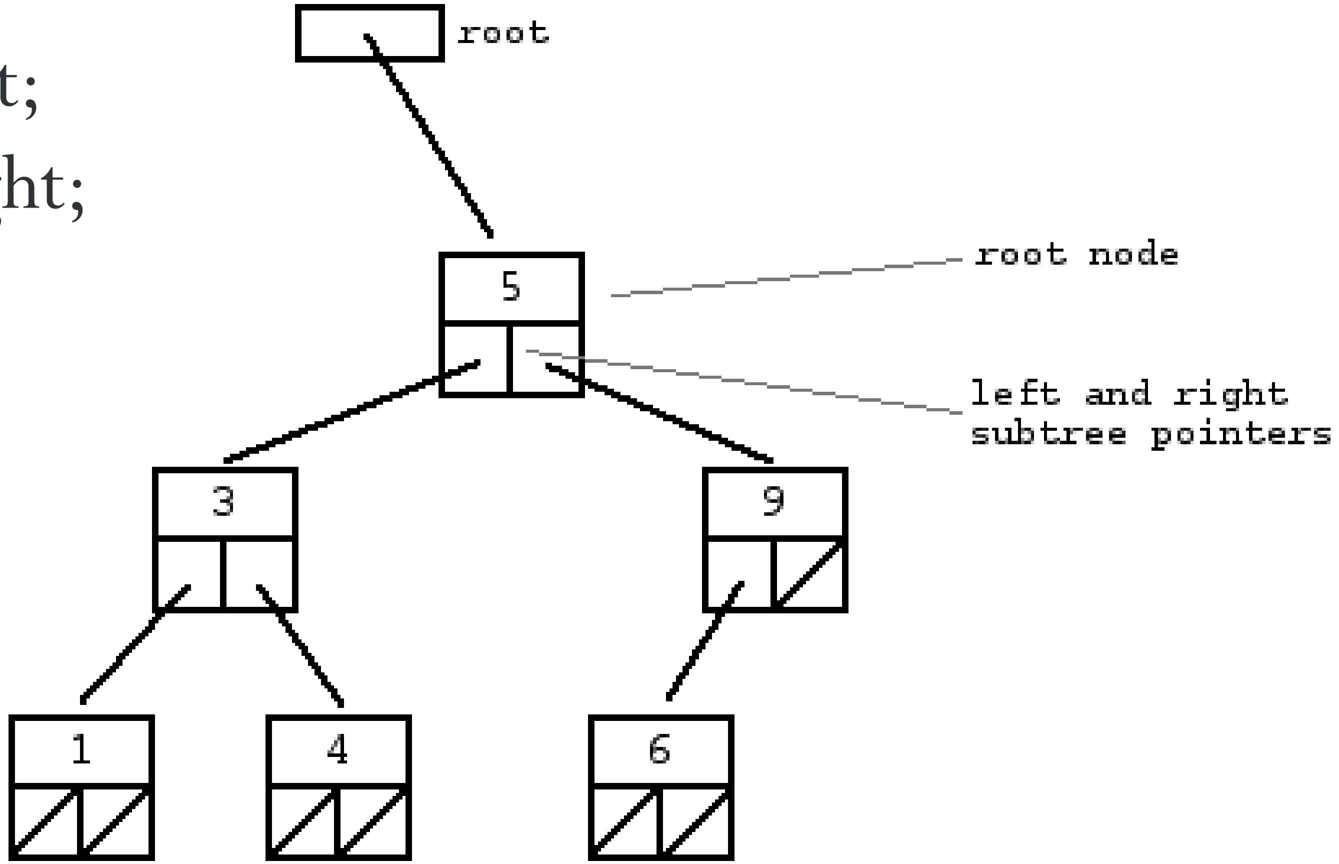
- A full binary tree is a binary tree in which each node has exactly zero or two children.
- A complete binary tree is a binary tree, which is completely filled, with the possible exception of the bottom level, which is filled from left to right.





# Binary Tree in C

```
struct node {
        int value;
        struct node * left;
        struct node * right;
    }
```

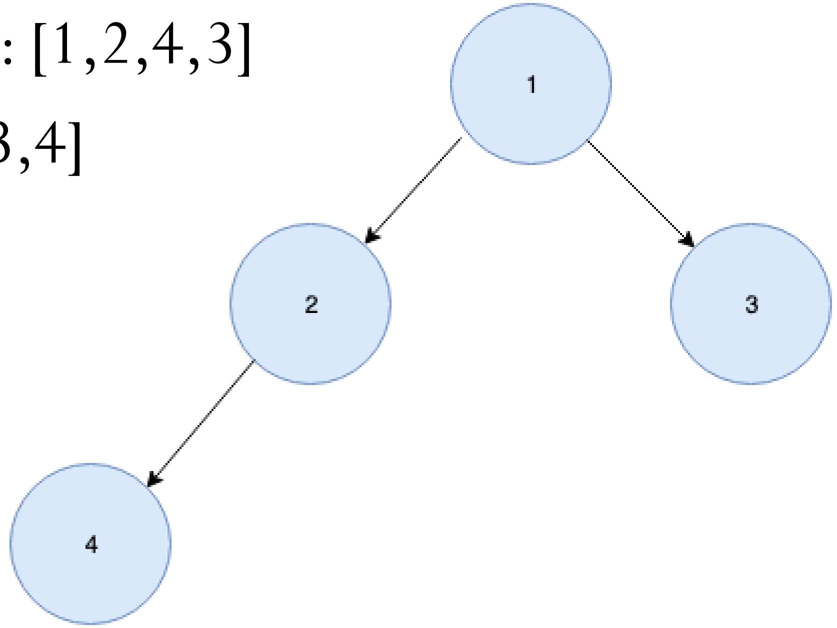


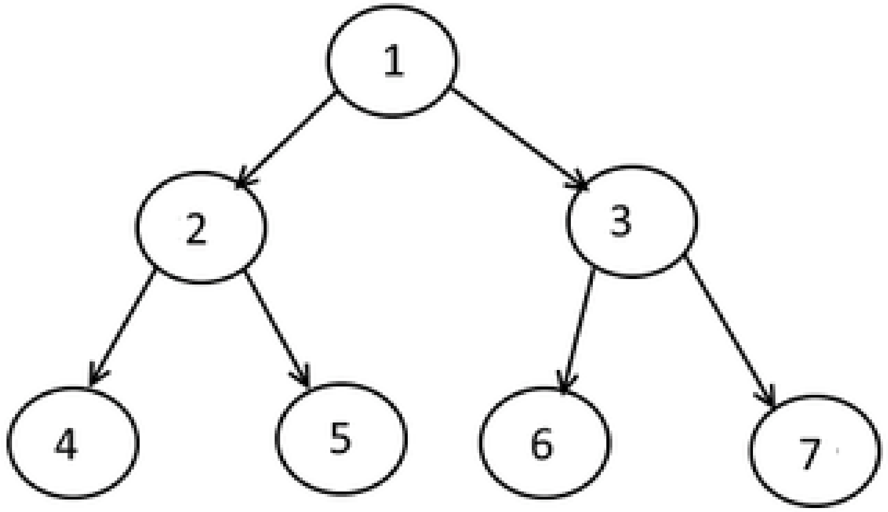
# Traversals

- A traversal is a process that visits all the nodes in the tree. Since a tree is a nonlinear data structure, there is no unique traversal. We will consider several traversal algorithms with we group in the following two kinds:-
  - depth-first traversal
  - breadth-first traversal
- There are three different types of depth-first traversals:-
  - **PreOrder traversal** - visit the parent first and then left and right children;
  - **InOrder traversal** - visit the left child, then the parent and the right child;
  - **PostOrder traversal** - visit left child, then the right child and then the parent;

# Traversals

- In-order traversal only in binary tree.
- The pre-order traversal is a form of the Depth-First-Search (DFS) traversal.
- The pre-order traversal is different from the Breadth First Search (BFS) traversal.
- Pre-Order: [1,2,4,3]
- BFS: [1,2,3,4]




Inorder Traversal: 4 2 5 1 6 3 7
Preorder Traversal: 1 2 4 5 3 6 7

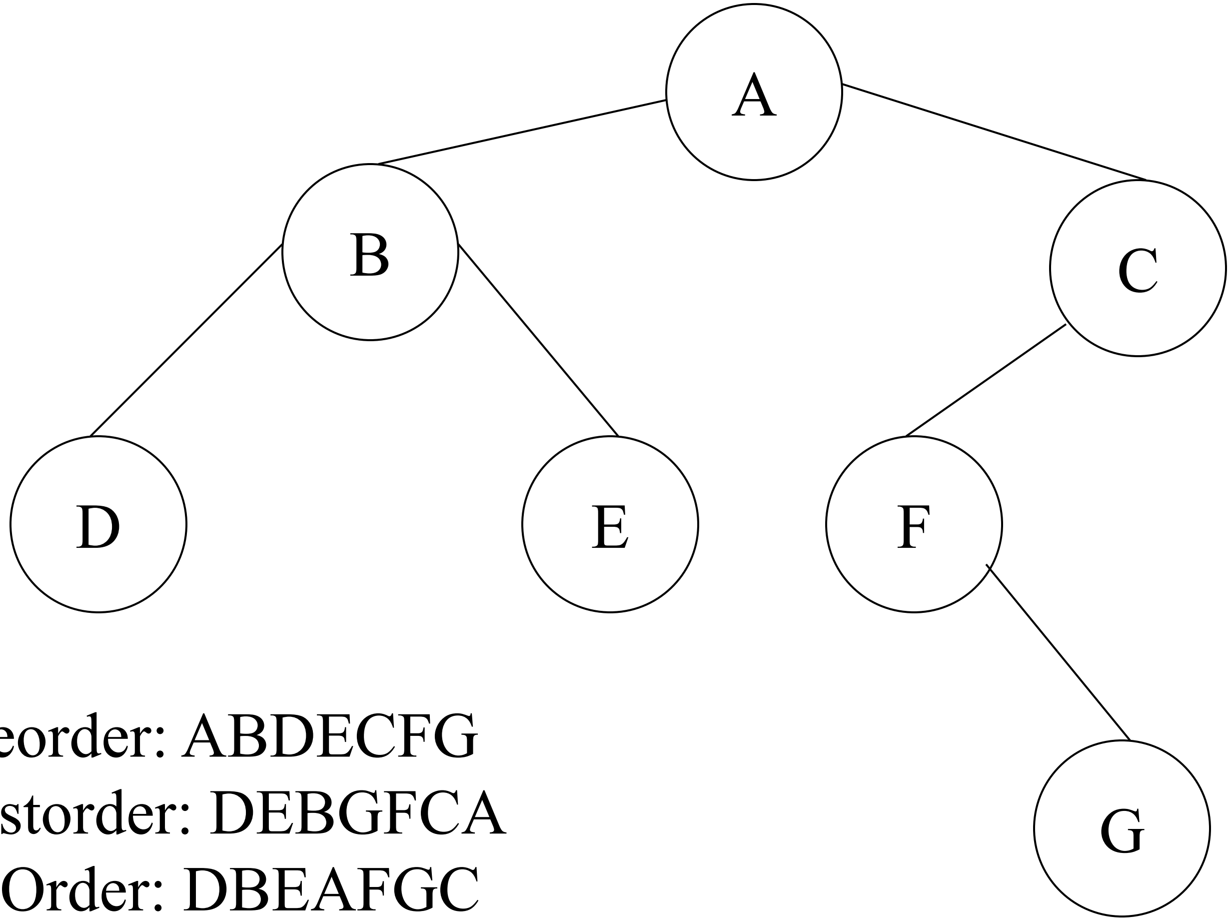
Breadth-First Search: 1 2 3 4 5 6 7
Depth-First Search: 1 2 4 5 3 6 7

Postorder Traversal: 4 5 2 6 7 3 1 (Left to Right)
Postorder Traversal: 7 6 3 5 4 2 1 (Right to Left) -- Reverse Order

# Reverse order Traversals

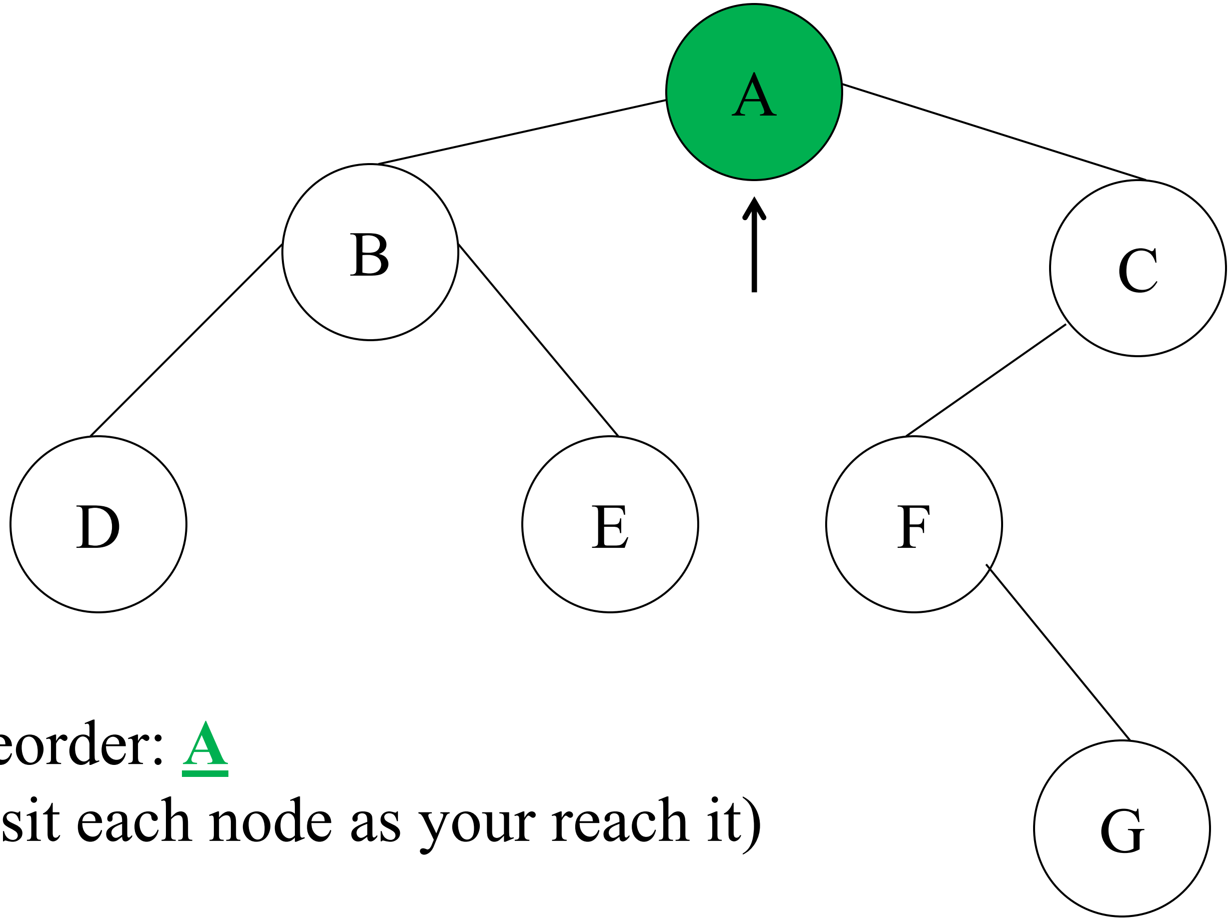
- The other traversals are the reverse of these three standard ones
  - the right subtree is traversed before the left subtree is traversed
- Reverse preorder: root, right subtree, left subtree
- Reverse inorder: right subtree, root, left subtree
- Reverse postorder: right subtree, left subtree, root

# Tree Traversals: An Example



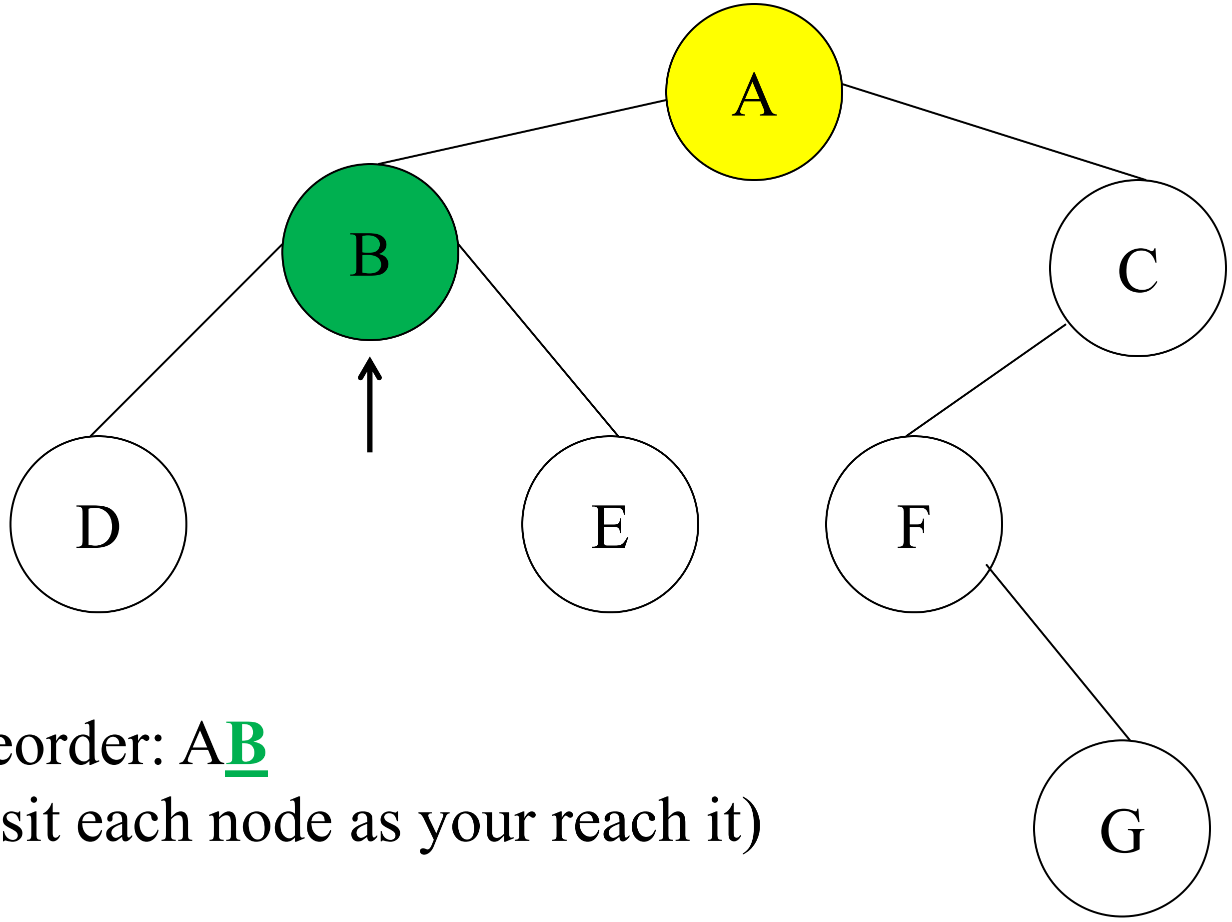

Preorder: ABDECFG
Postorder: DEBGFCA
In Order: DBEAFGC

# Tree Traversals: An Example



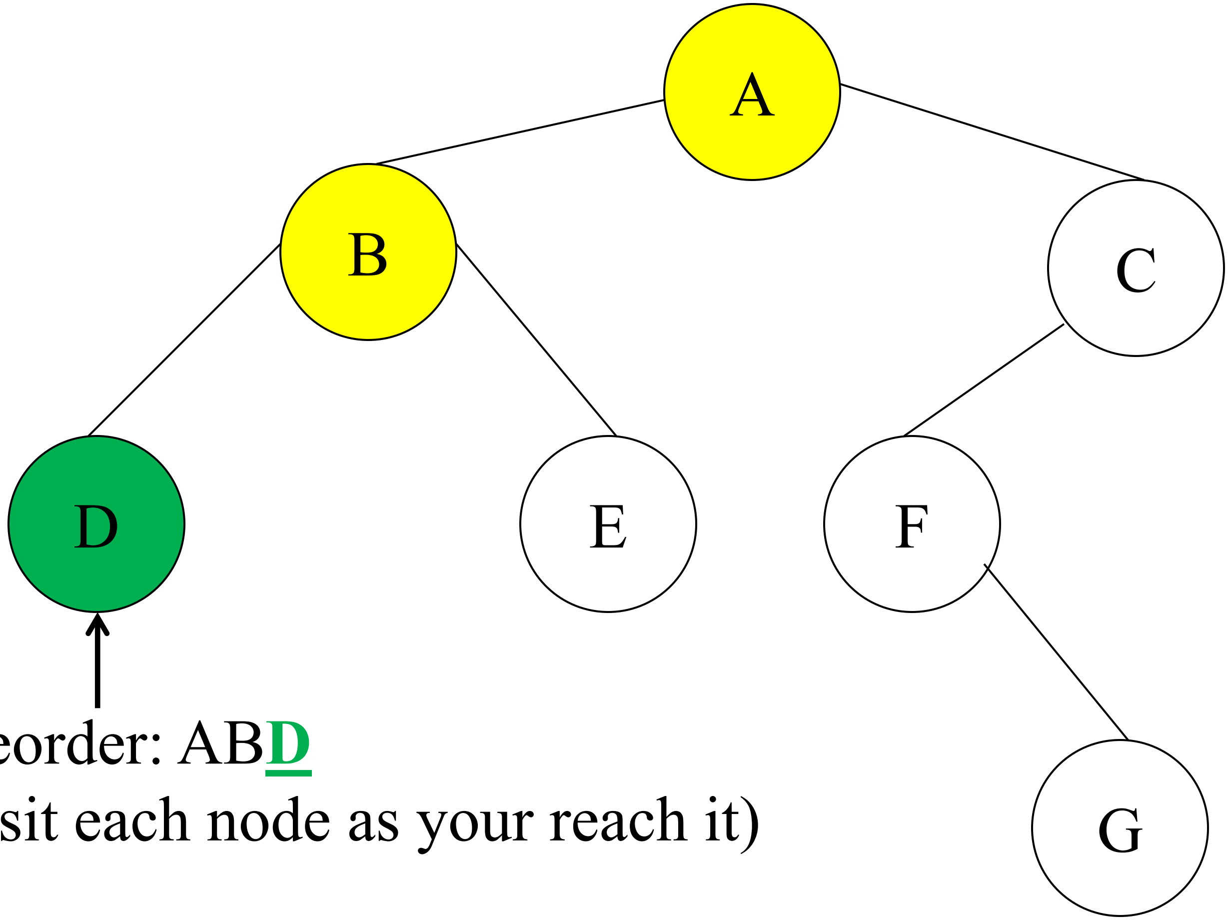

Preorder: **A**
(visit each node as your reach it)

# Tree Traversals: An Example



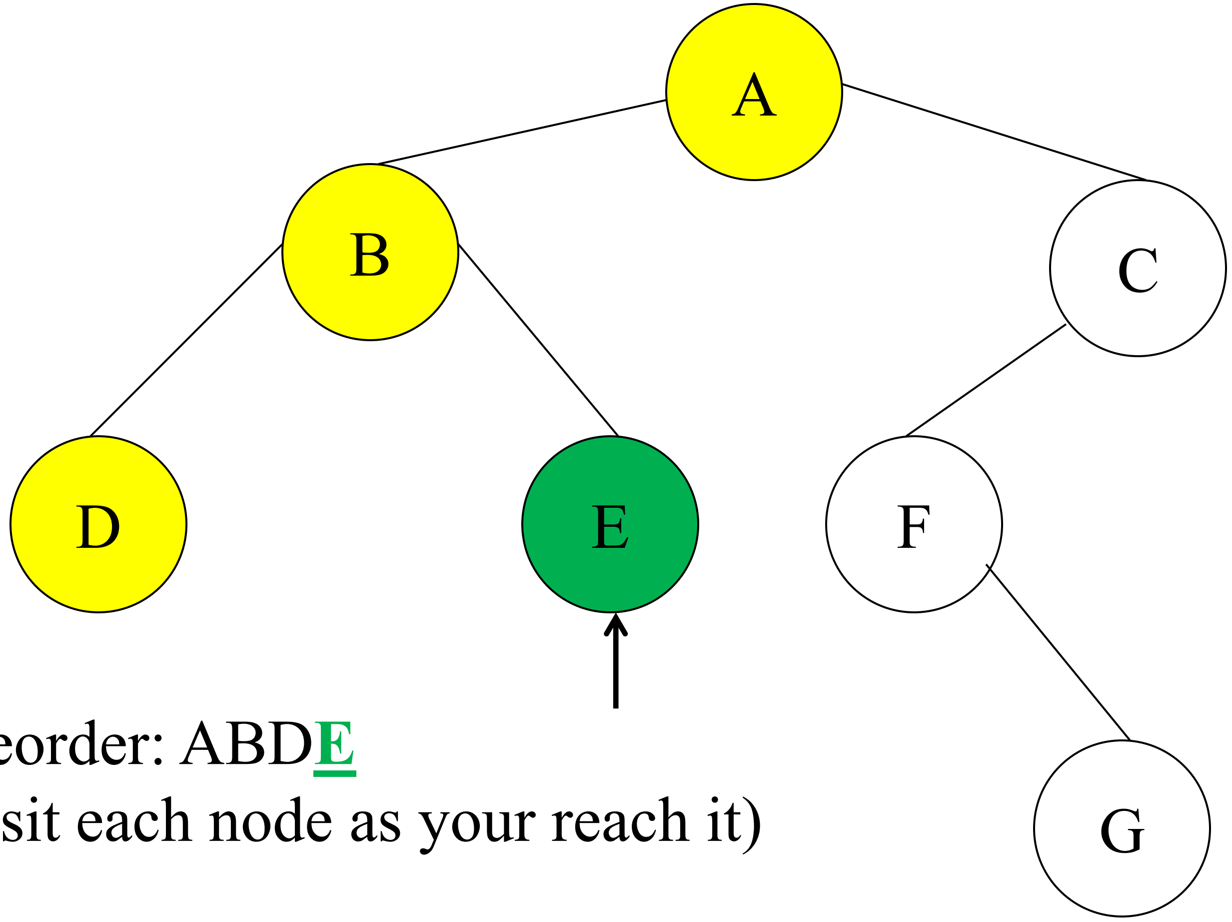

Preorder: A**B**
(visit each node as your reach it)

# Tree Traversals: An Example



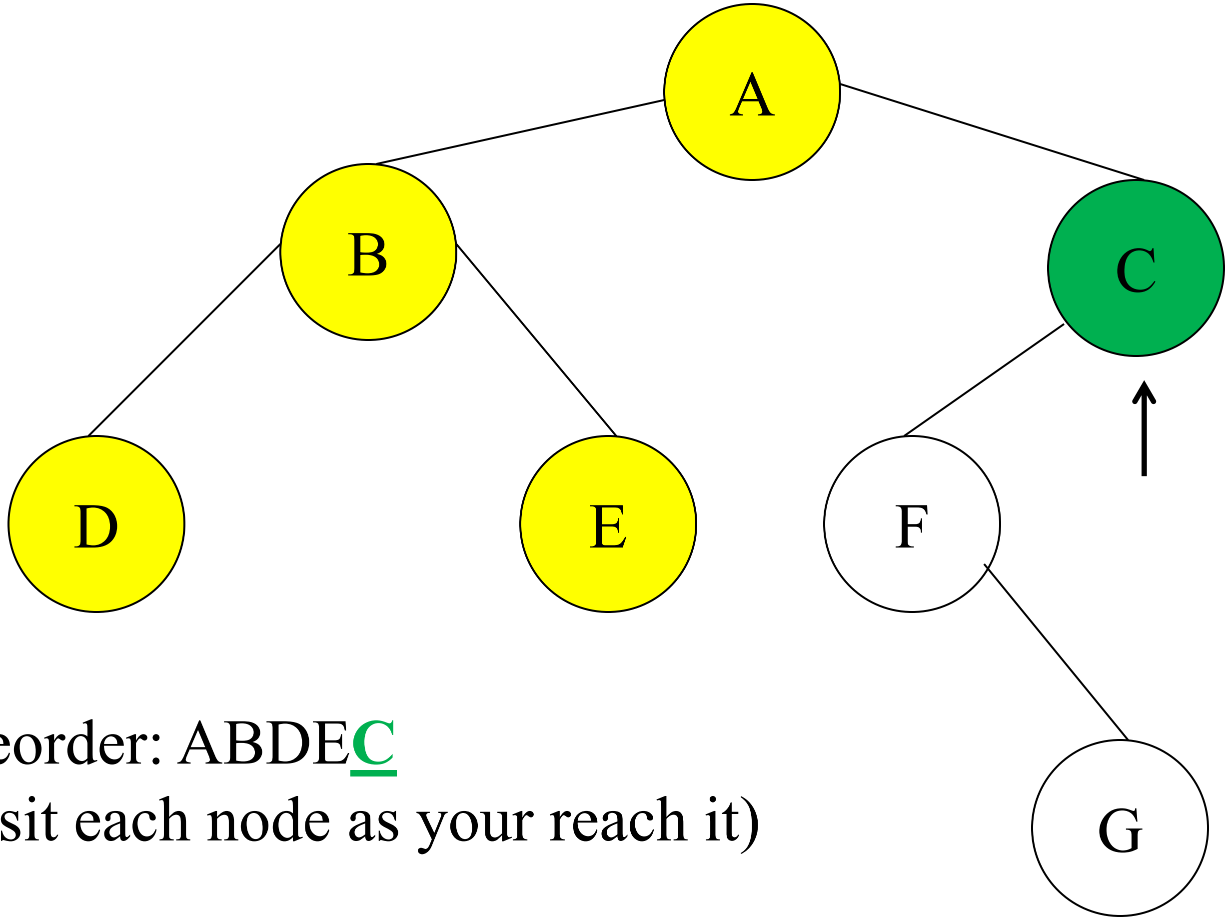

Preorder: AB**D**
(visit each node as your reach it)

# Tree Traversals: An Example



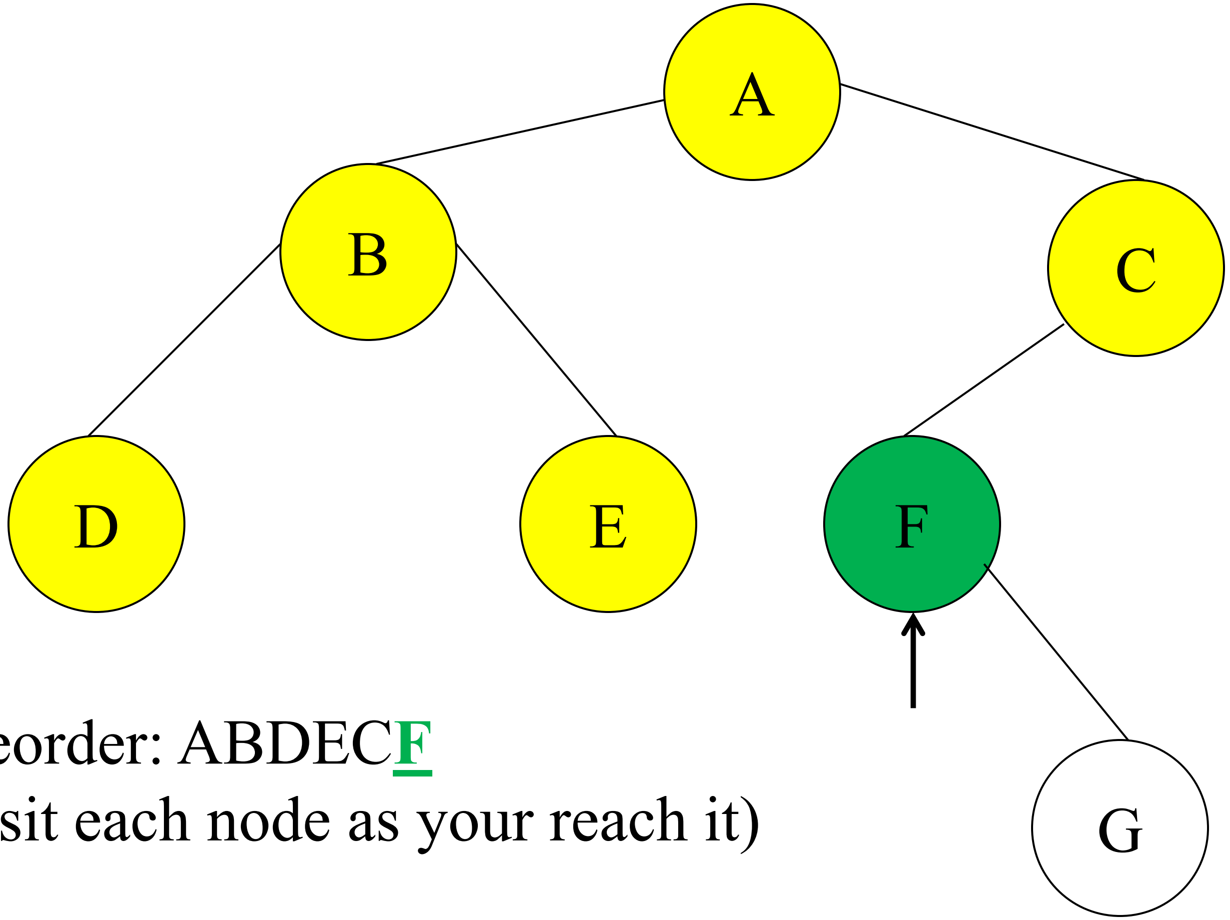

Preorder: ABD**E**
(visit each node as your reach it)

# Tree Traversals: An Example



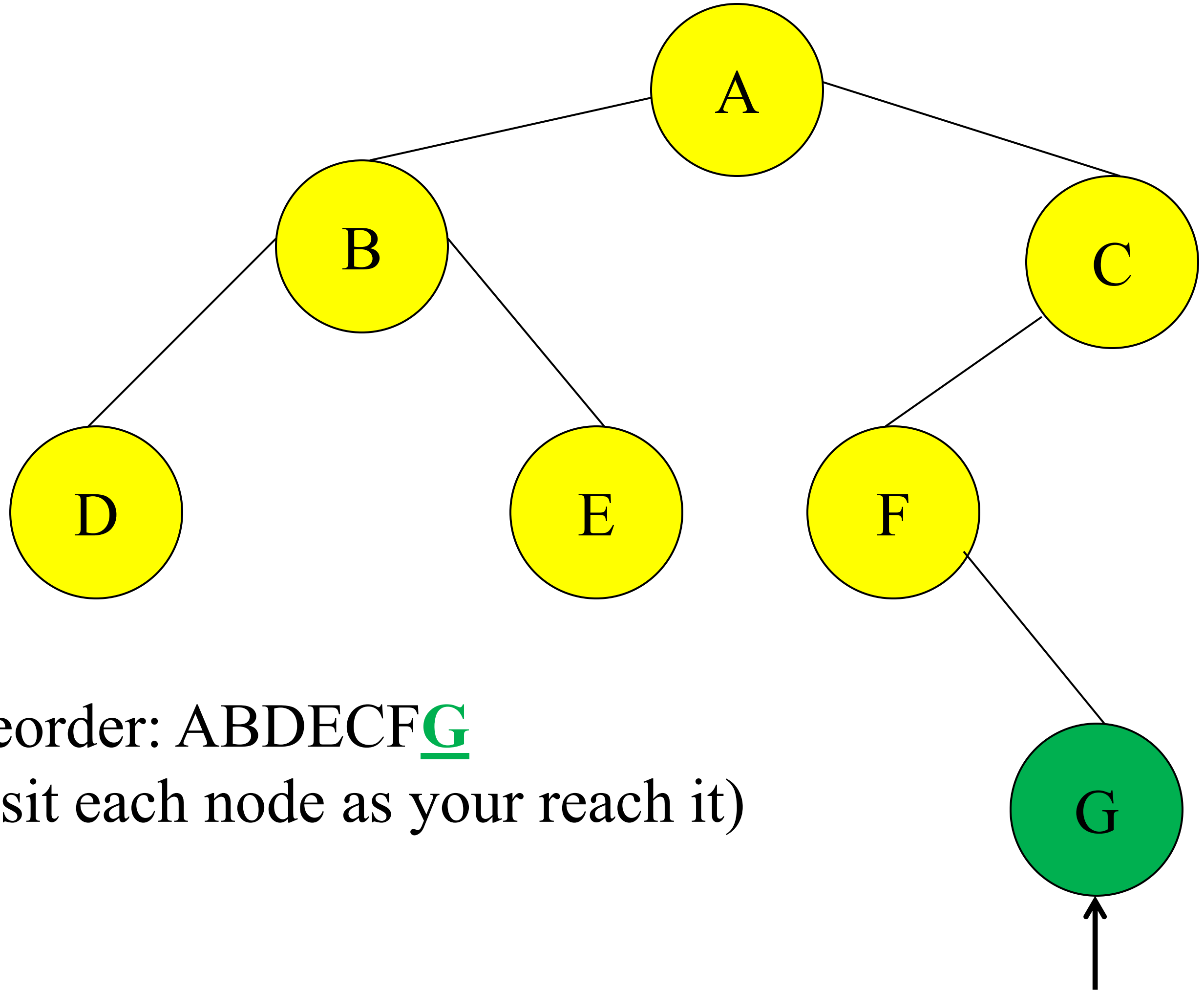

Preorder: ABDE**C**
(visit each node as your reach it)

# Tree Traversals: An Example



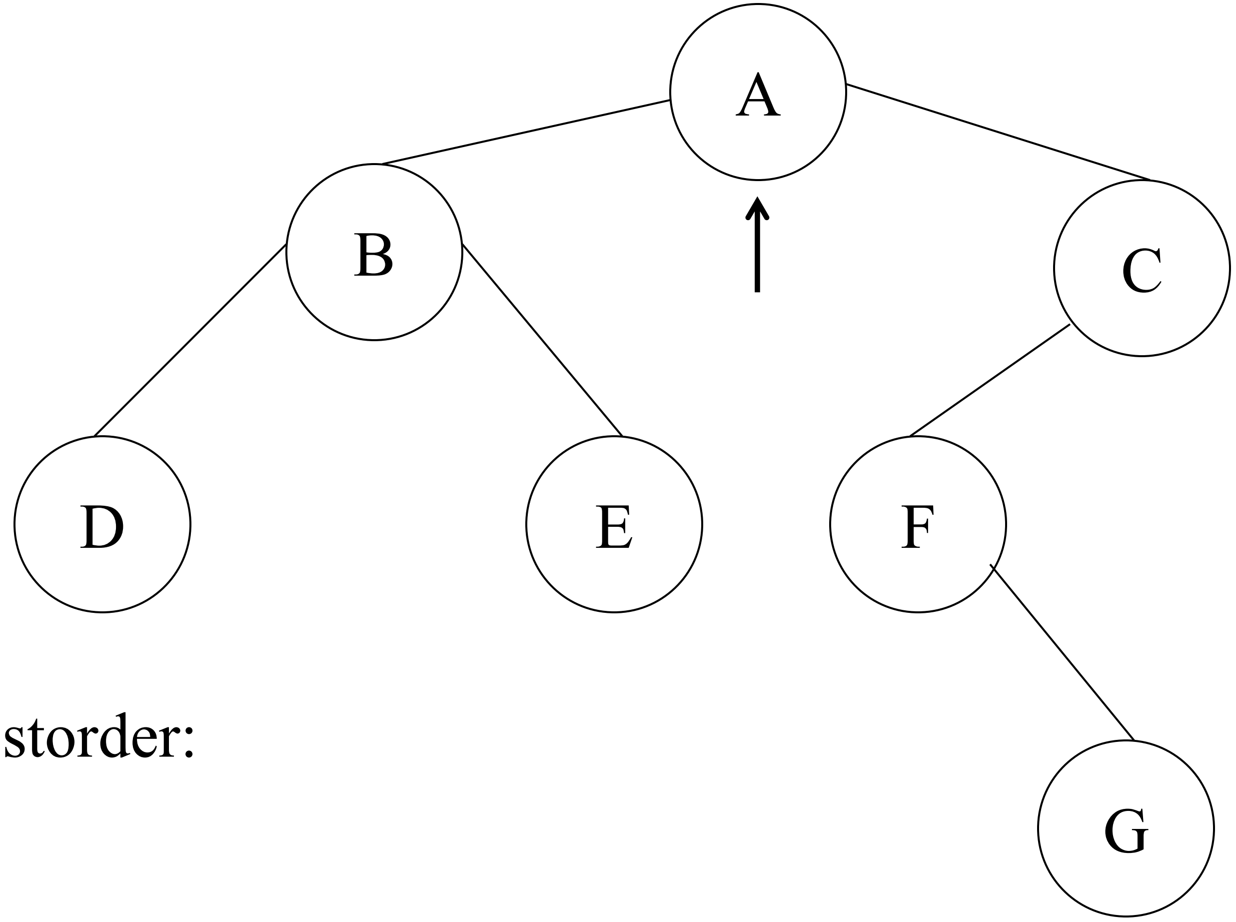

Preorder: ABDEC**F**
(visit each node as your reach it)

# Tree Traversals: An Example



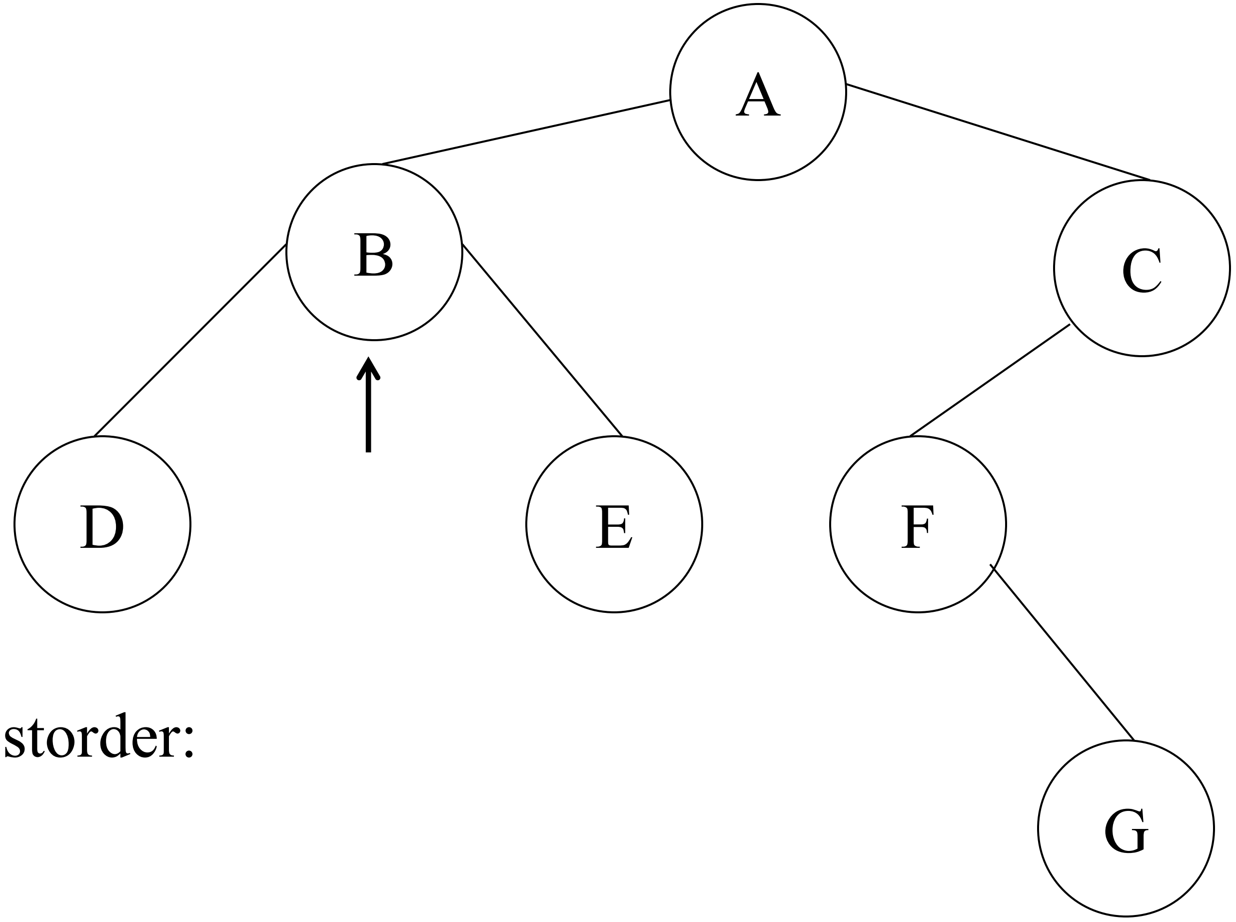

Preorder: ABDECF**G**
(visit each node as your reach it)

# Tree Traversals: An Example



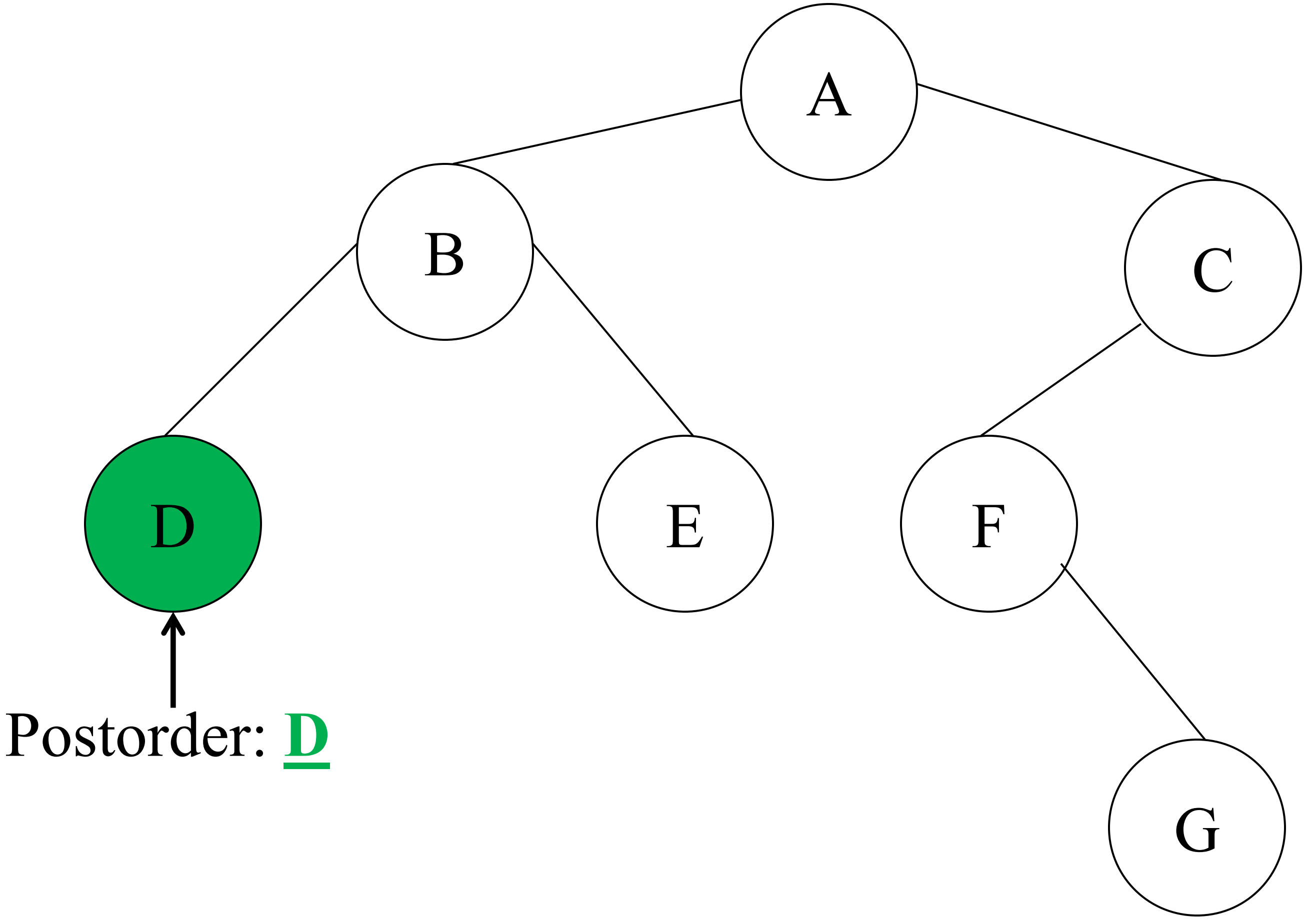

Postorder:

# Tree Traversals: An Example

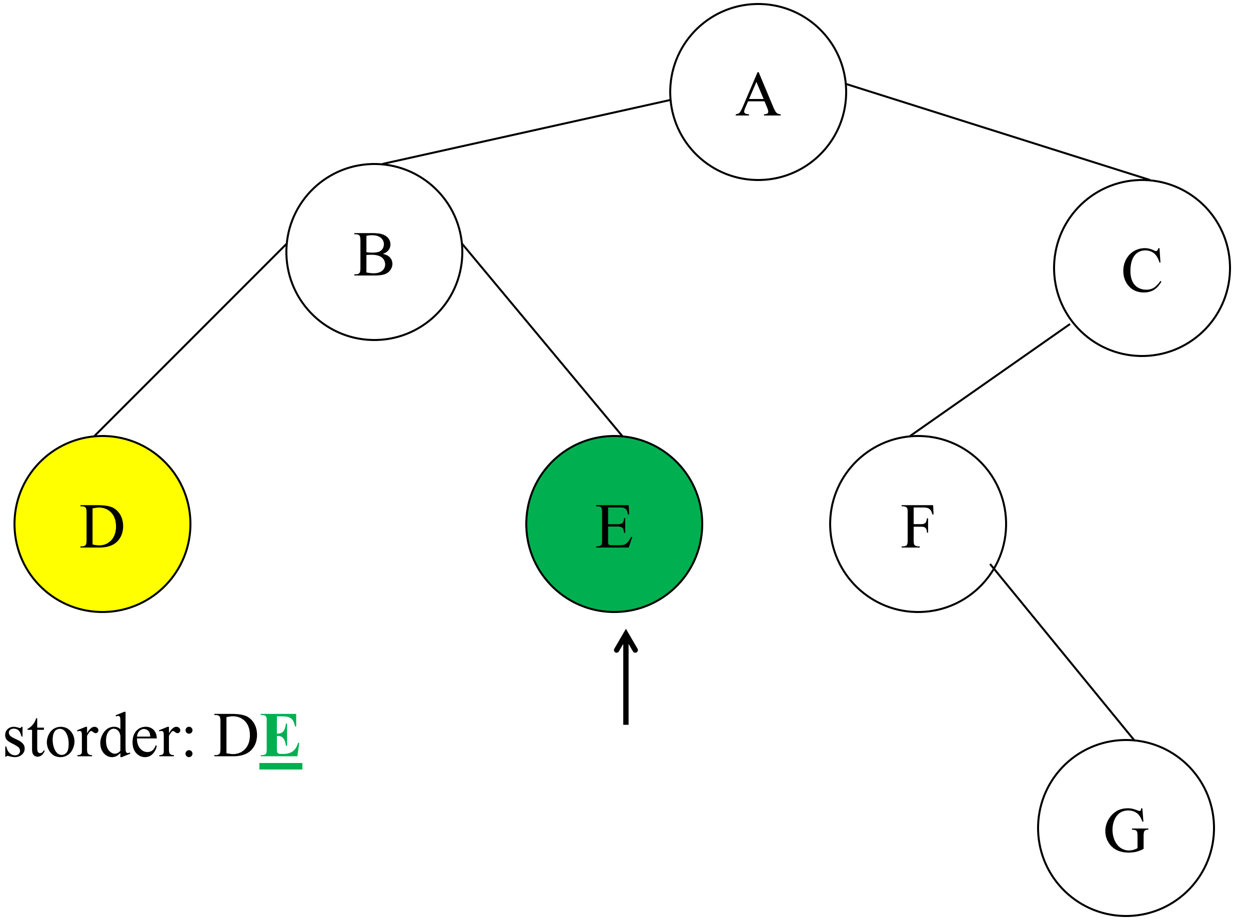



Postorder:

# Tree Traversals: An Example

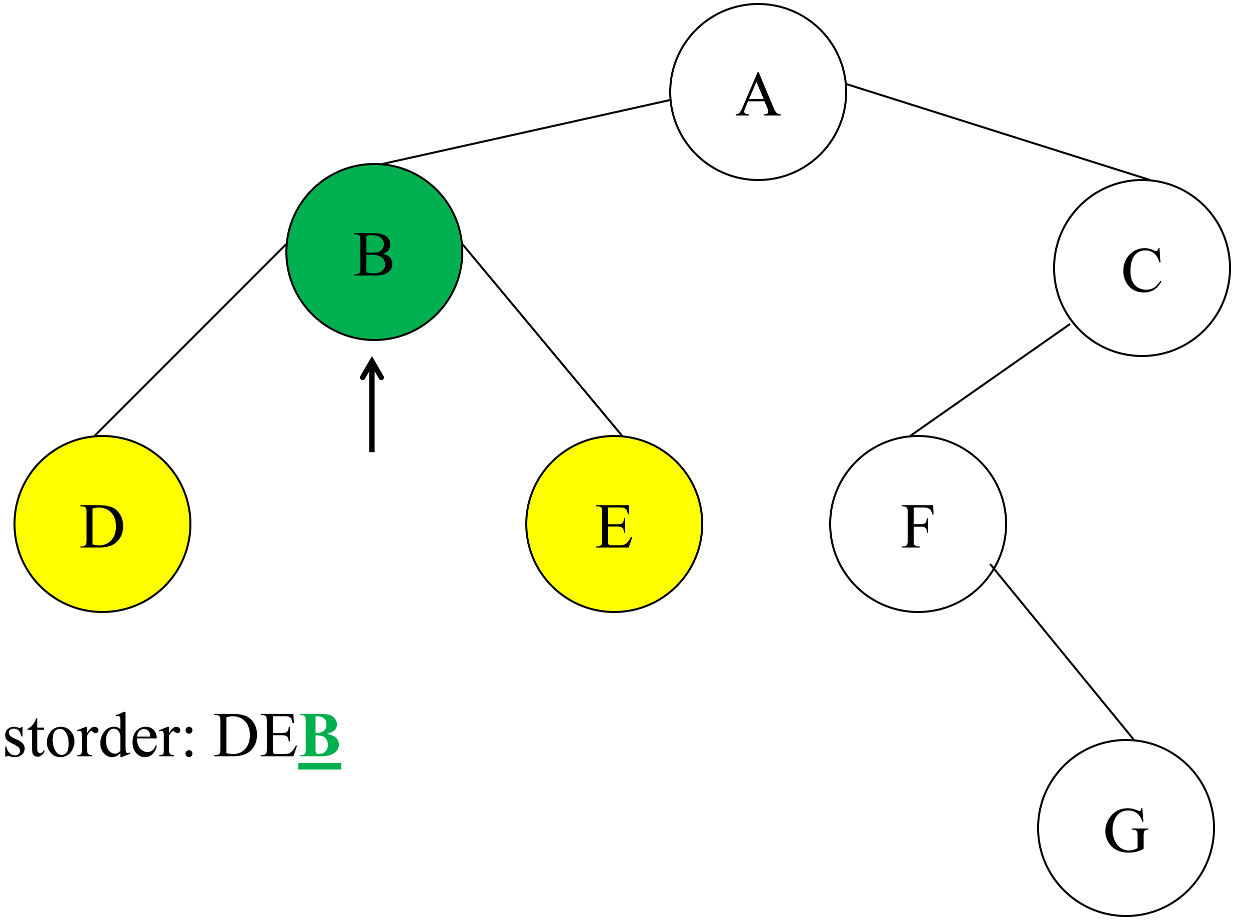


# Tree Traversals: An Example



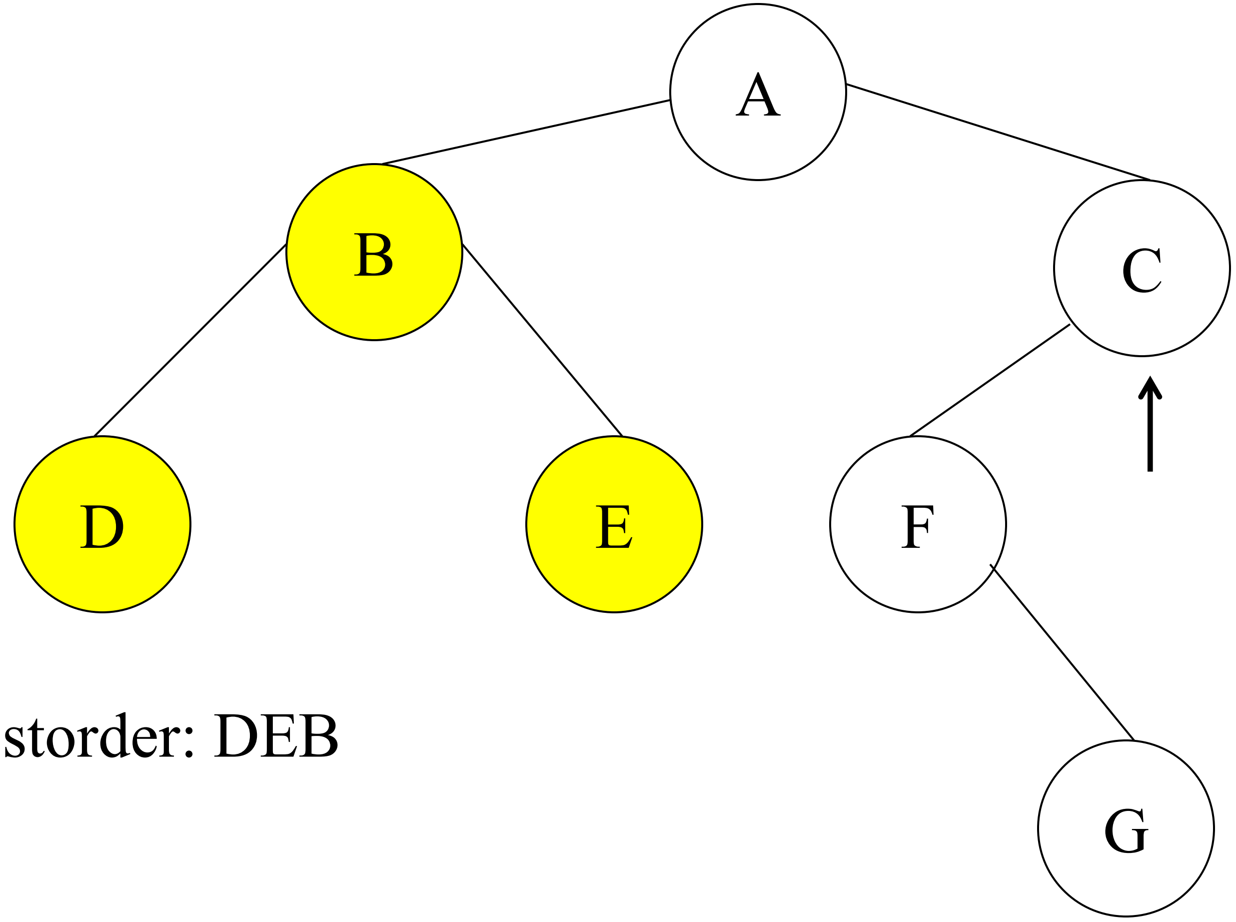

Postorder: D**E**

# Tree Traversals: An Example



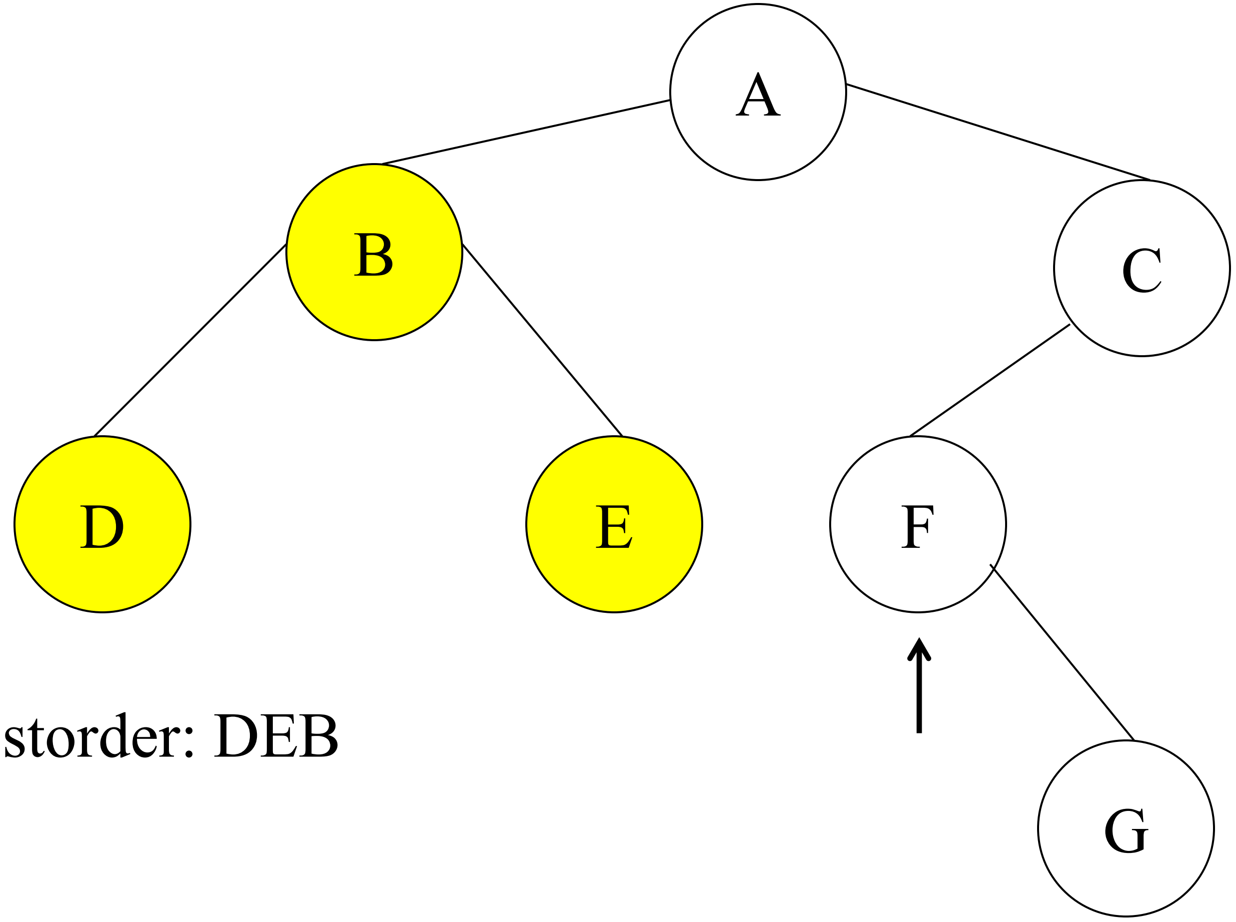

Postorder: DE**B**

# Tree Traversals: An Example



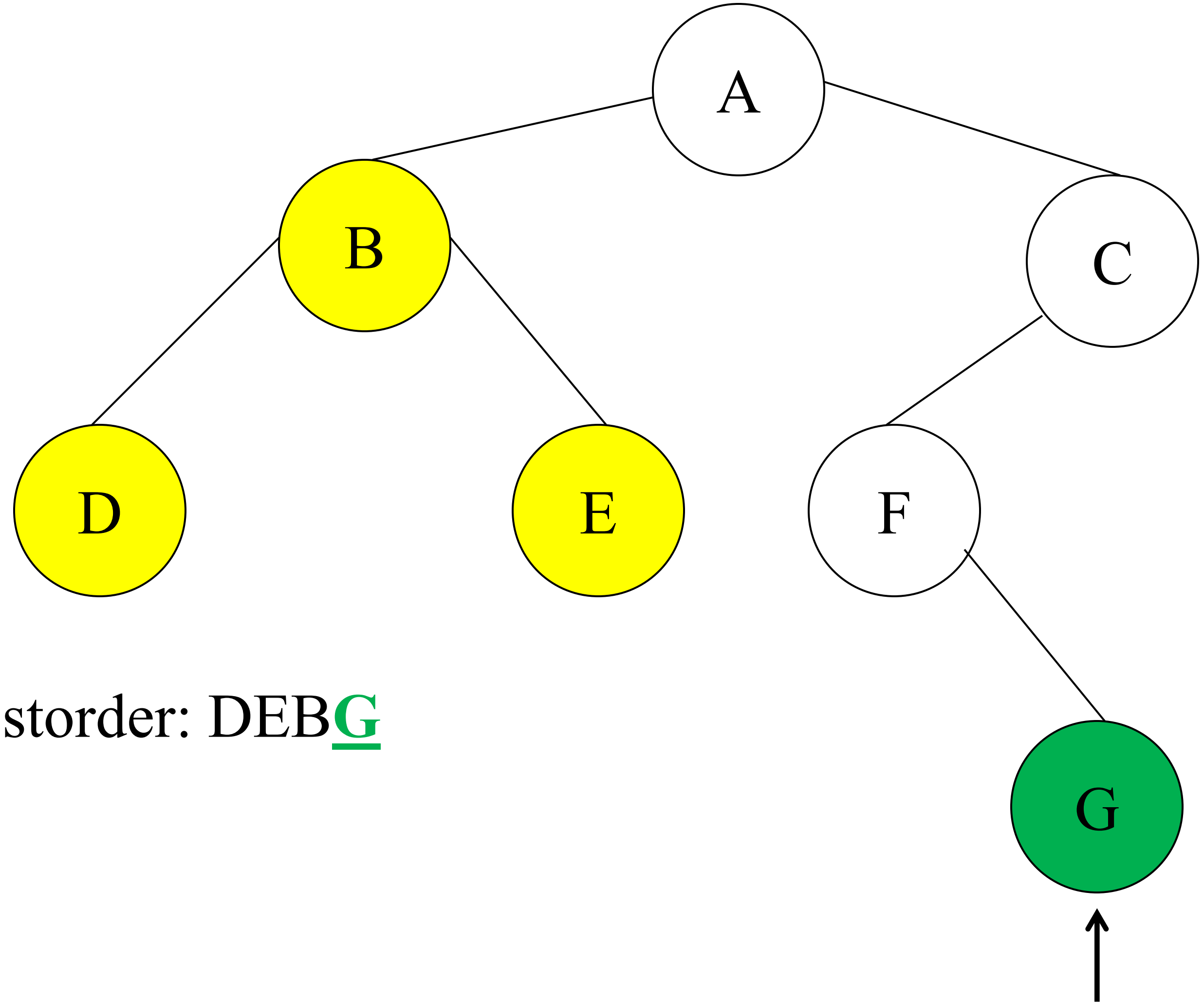

Postorder: DEB

# Tree Traversals: An Example



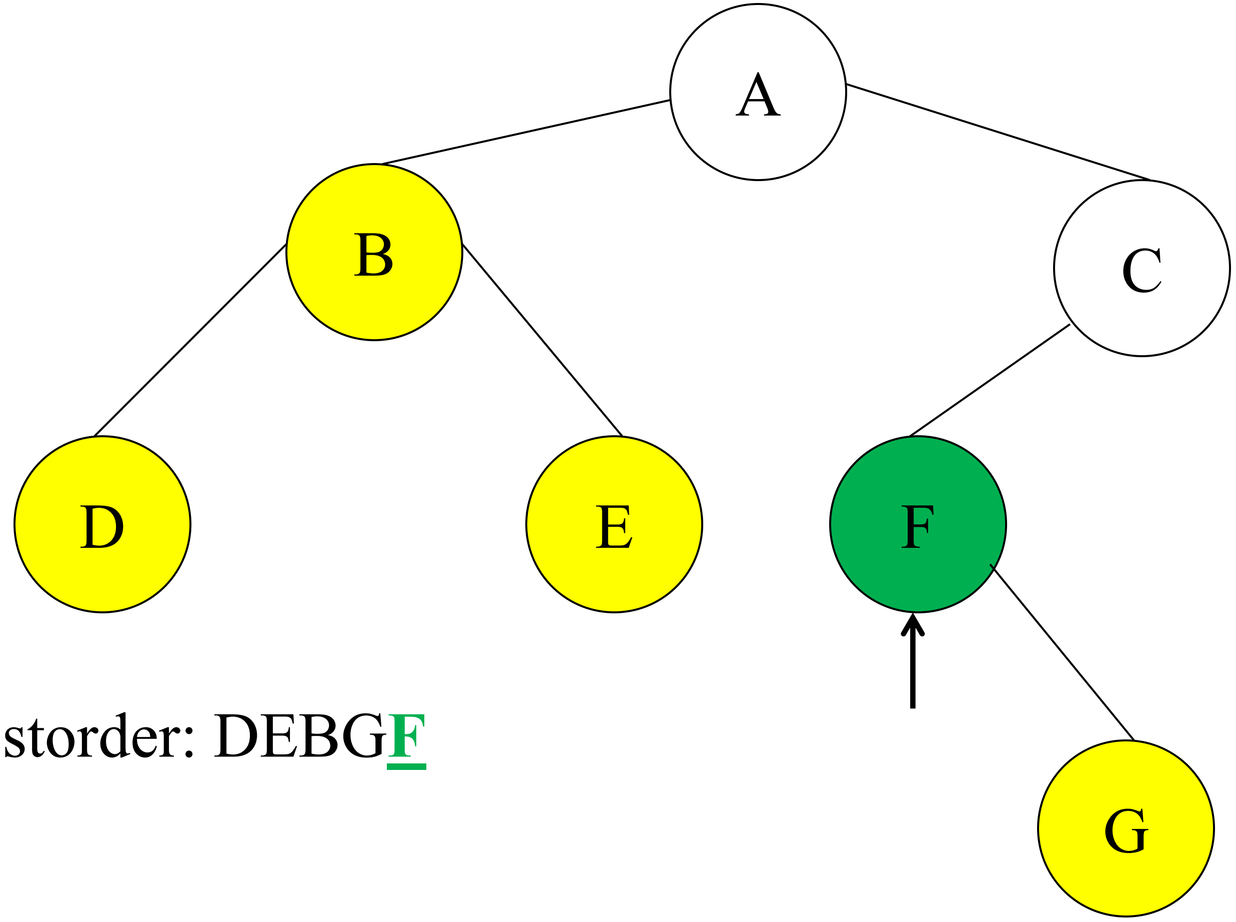

Postorder: DEB

# Tree Traversals: An Example



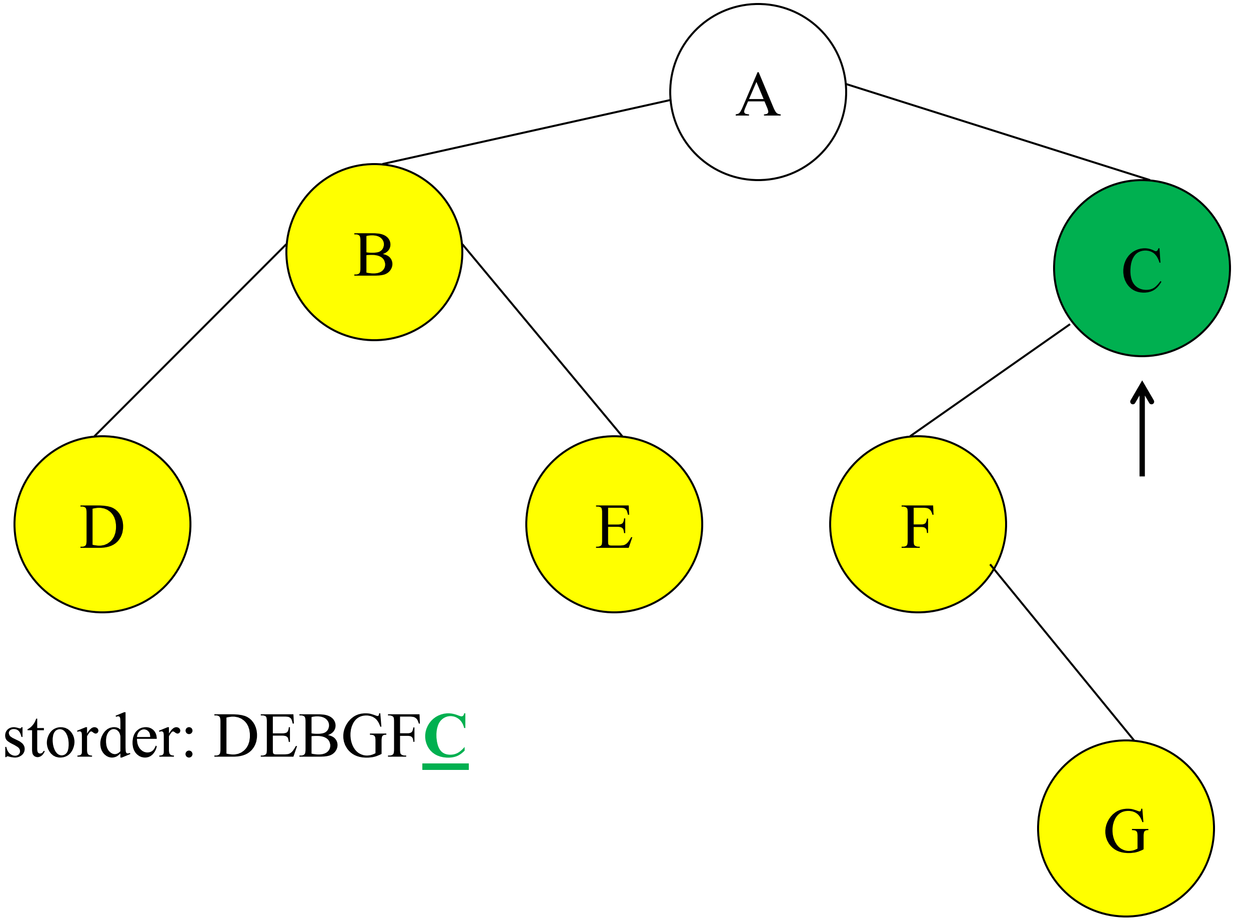

Postorder: DEB**G**

# Tree Traversals: An Example



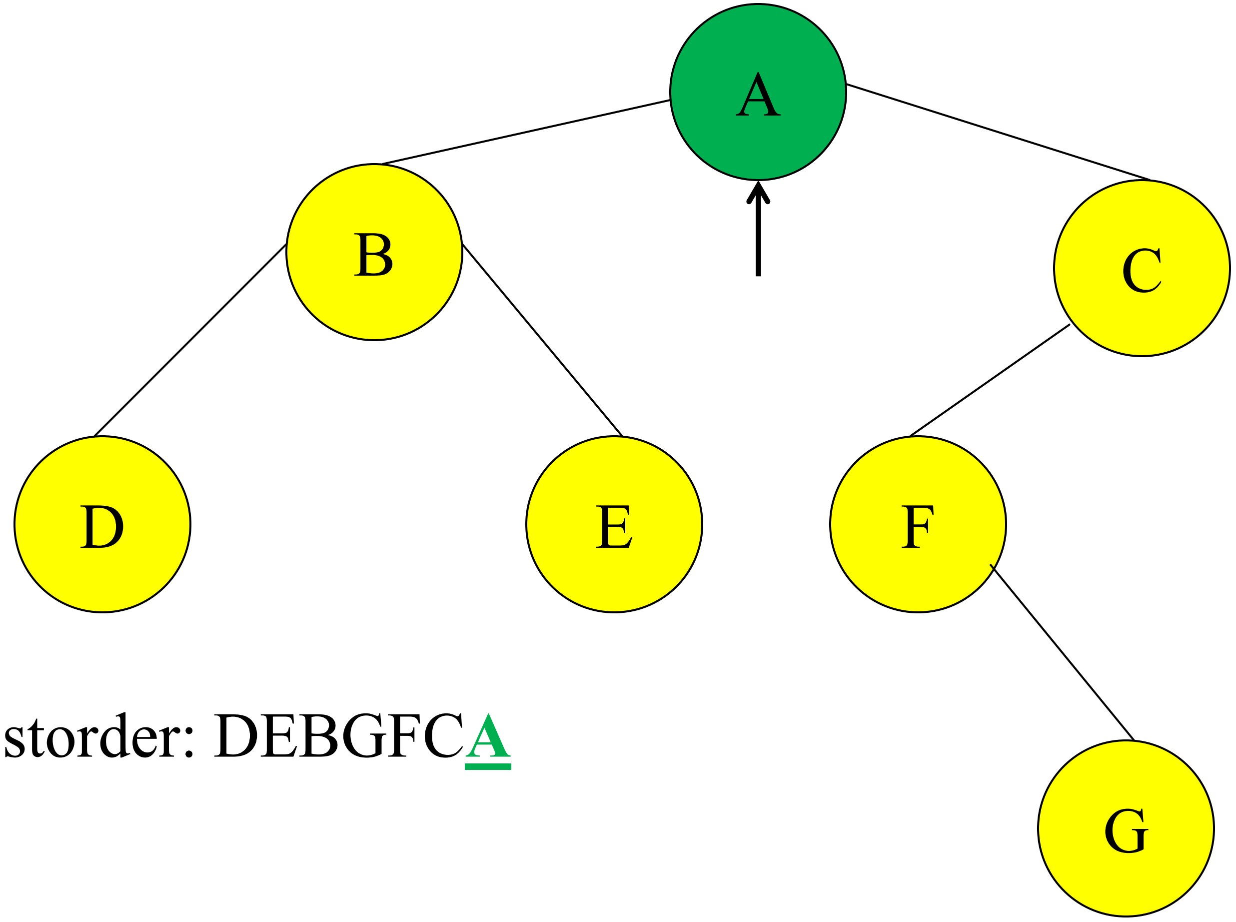

Postorder: DEBG**F**

# Tree Traversals: An Example



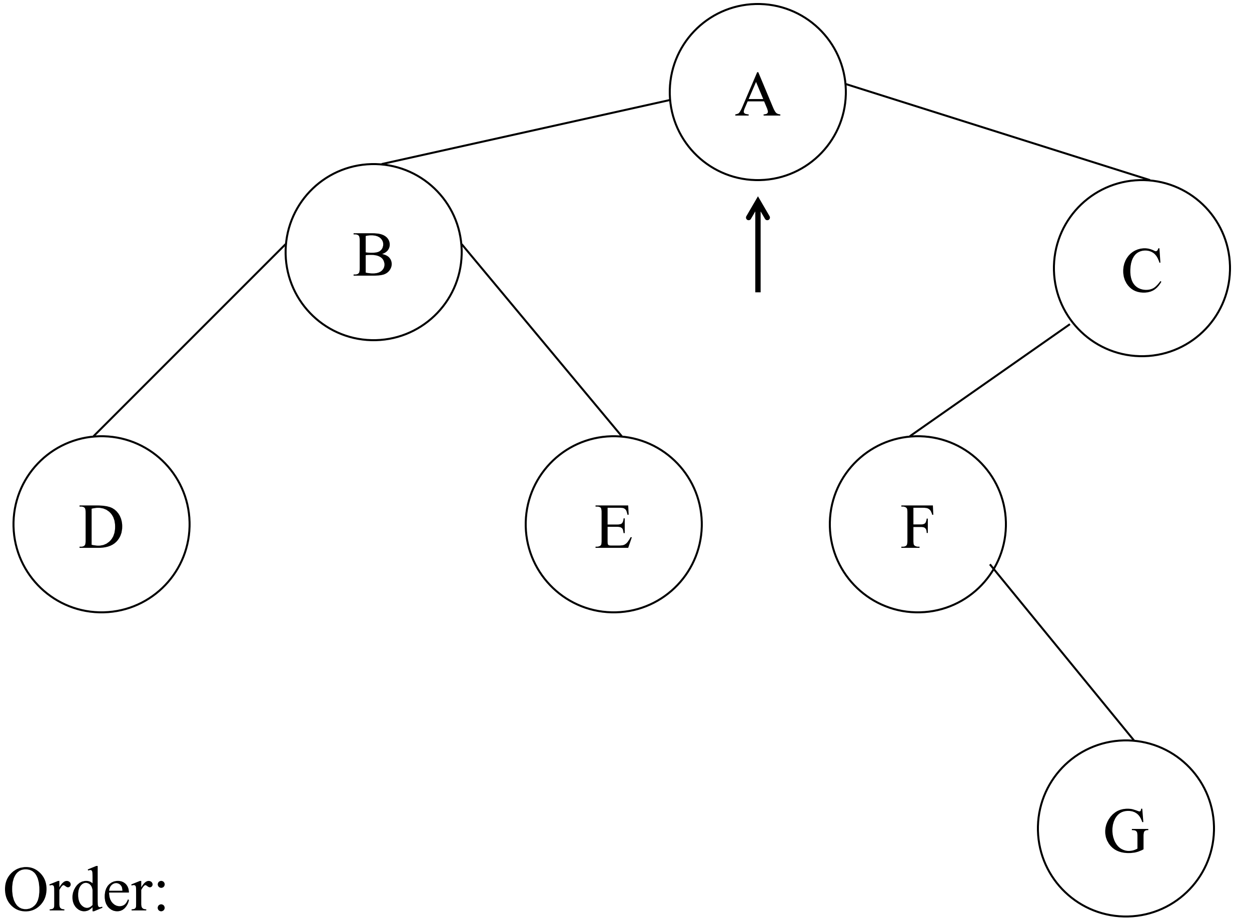

Postorder: DEBGF**C**

# Tree Traversals: An Example



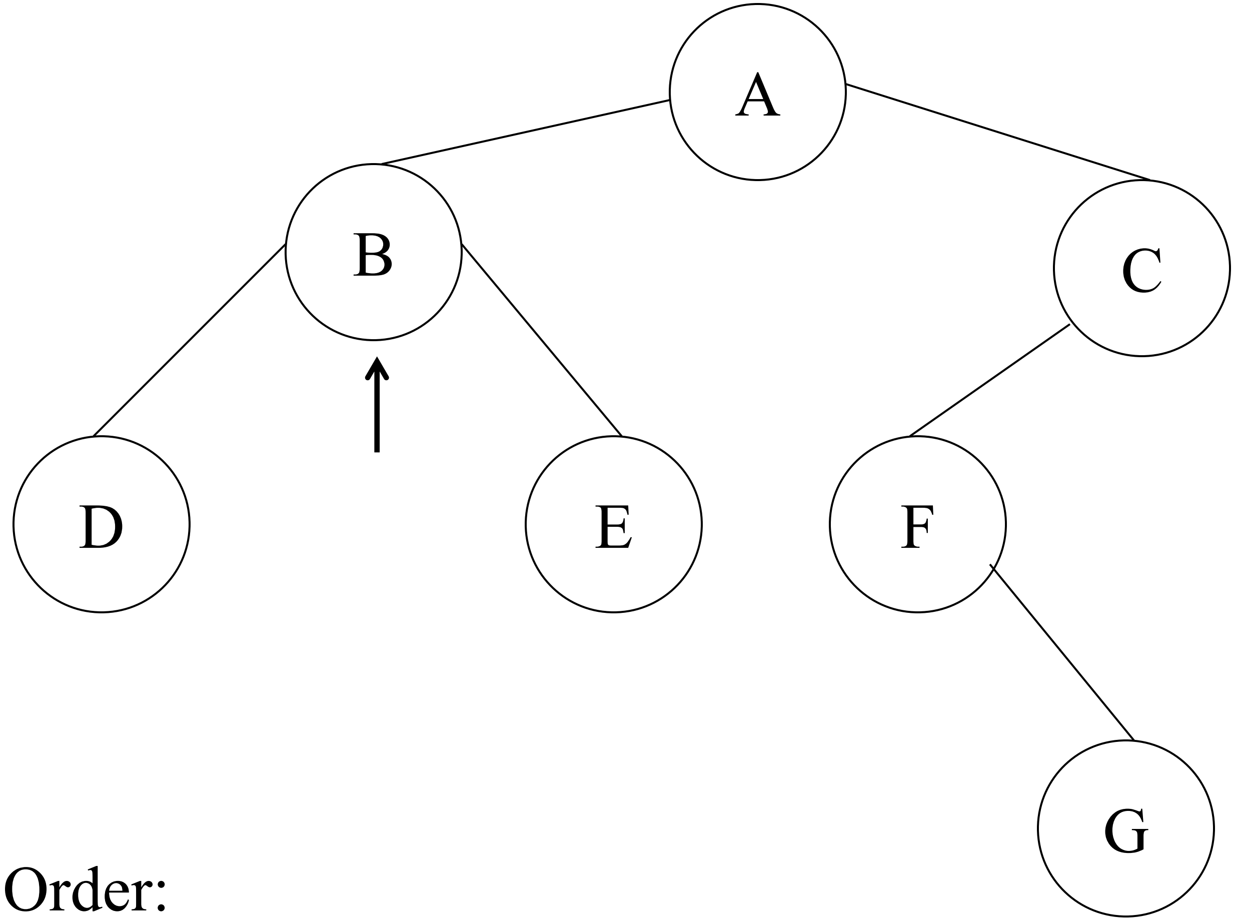

Postorder: DEBGFC**A**

# Tree Traversals: An Example



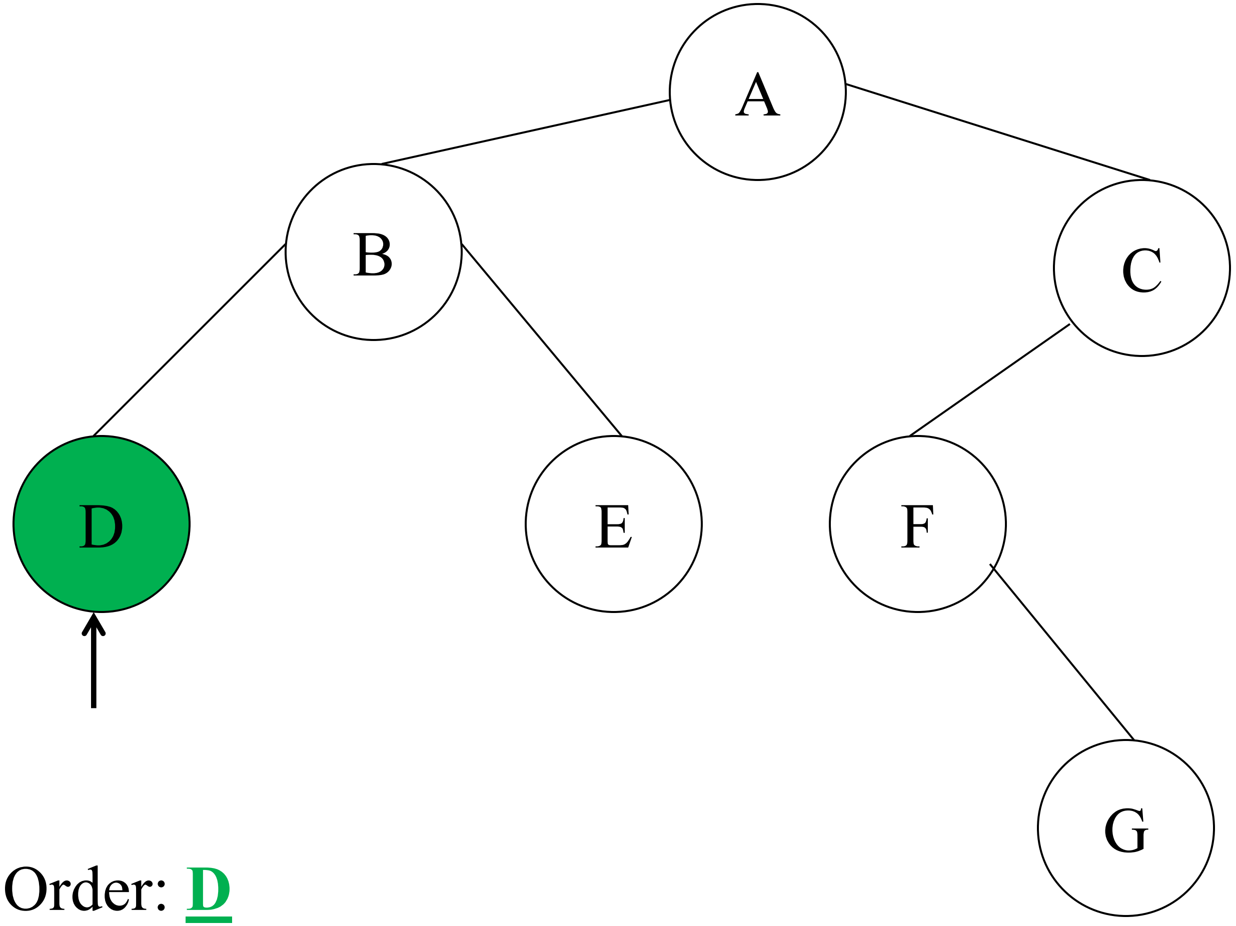

In Order:

# Tree Traversals: An Example



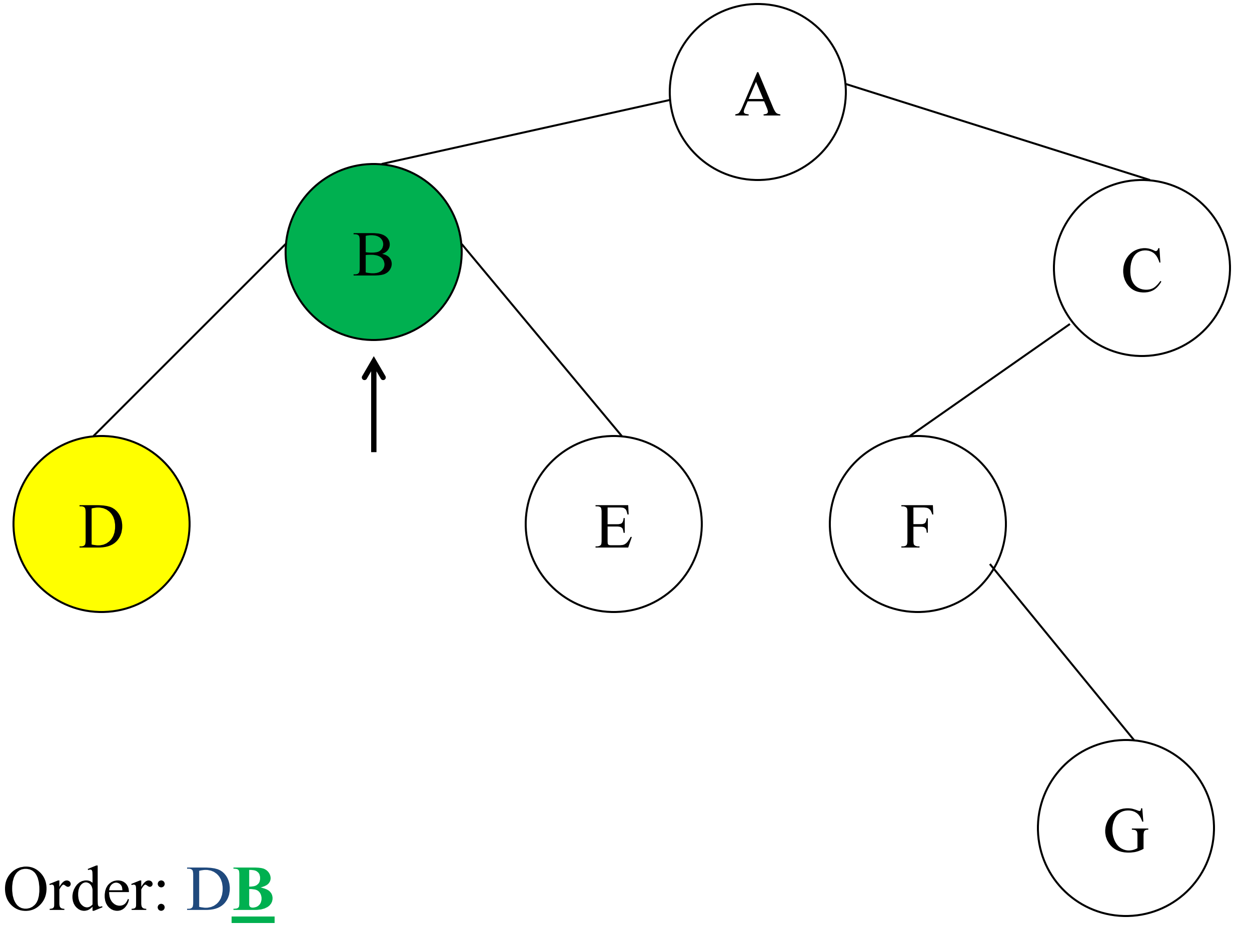

In Order:

# Tree Traversals: An Example



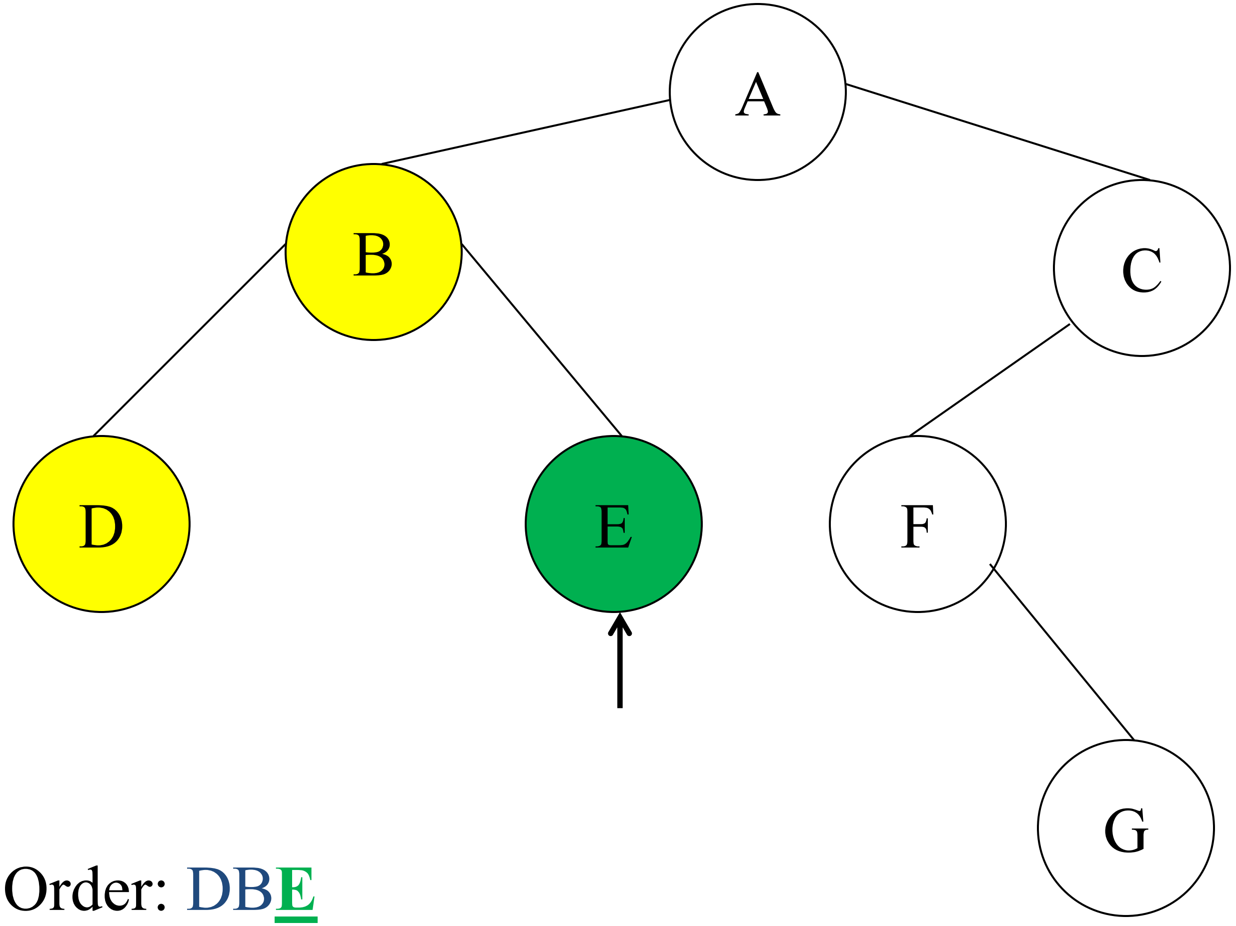

In Order: **D**

# Tree Traversals: An Example



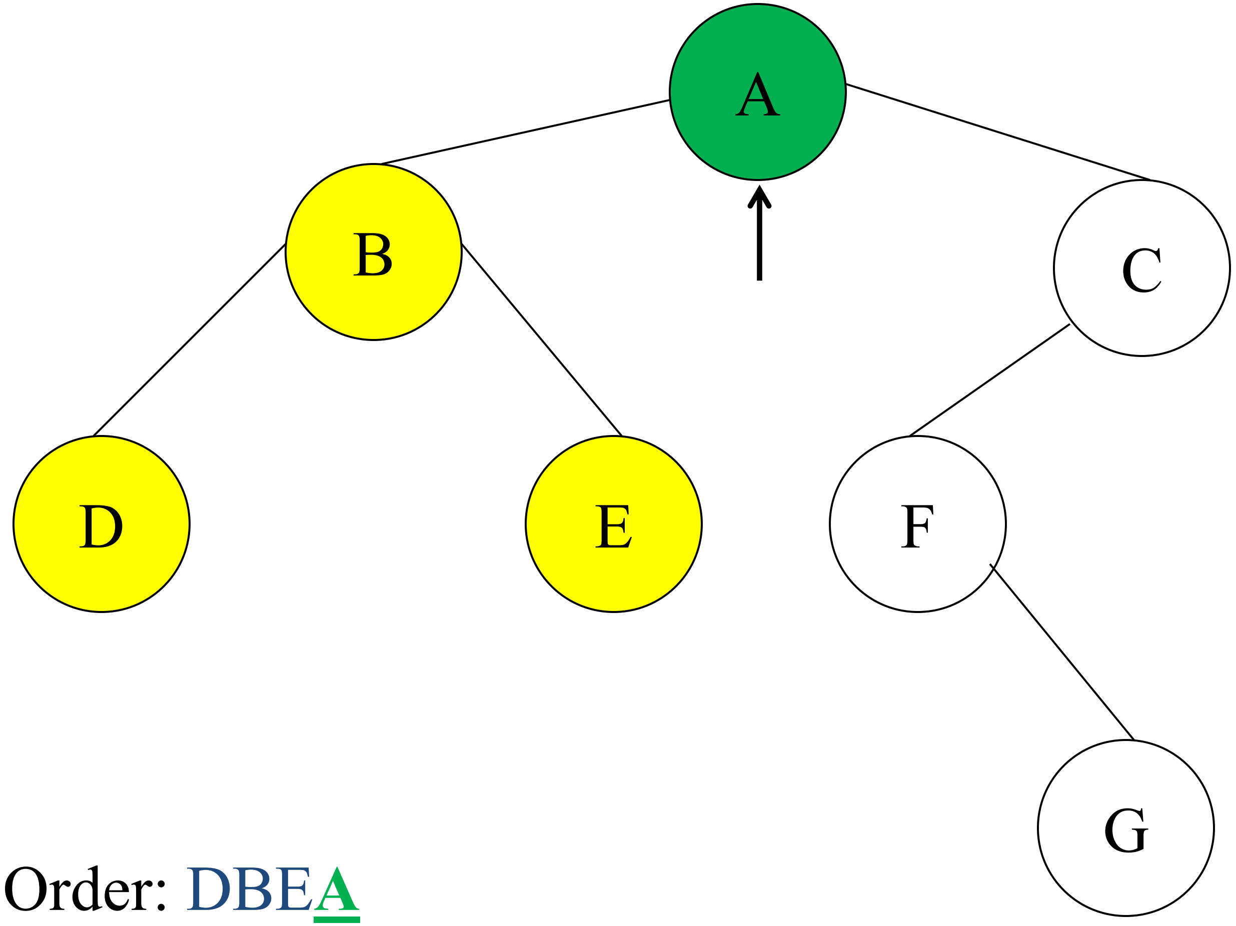

In Order: D**B**

# Tree Traversals: An Example



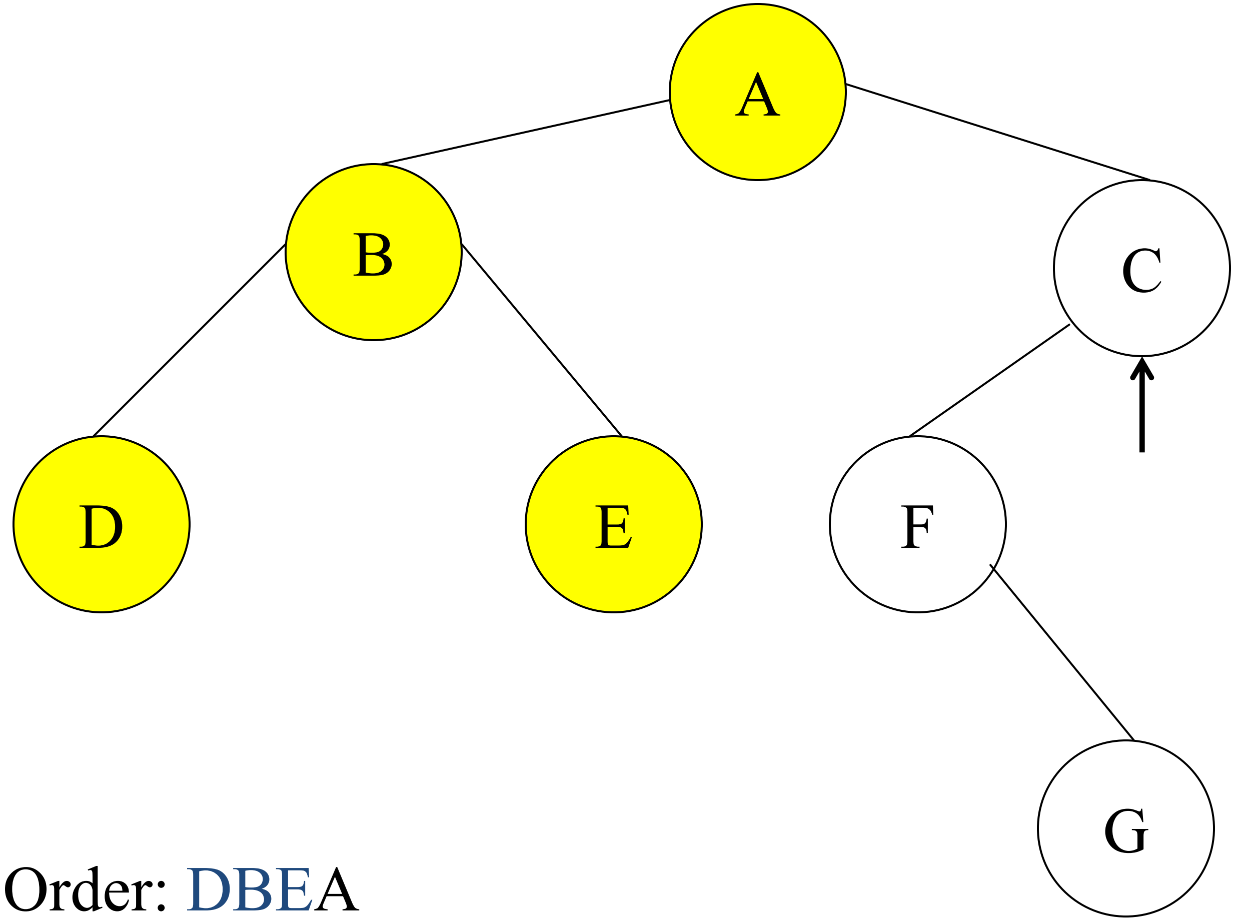

In Order: DB**E**

# Tree Traversals: An Example



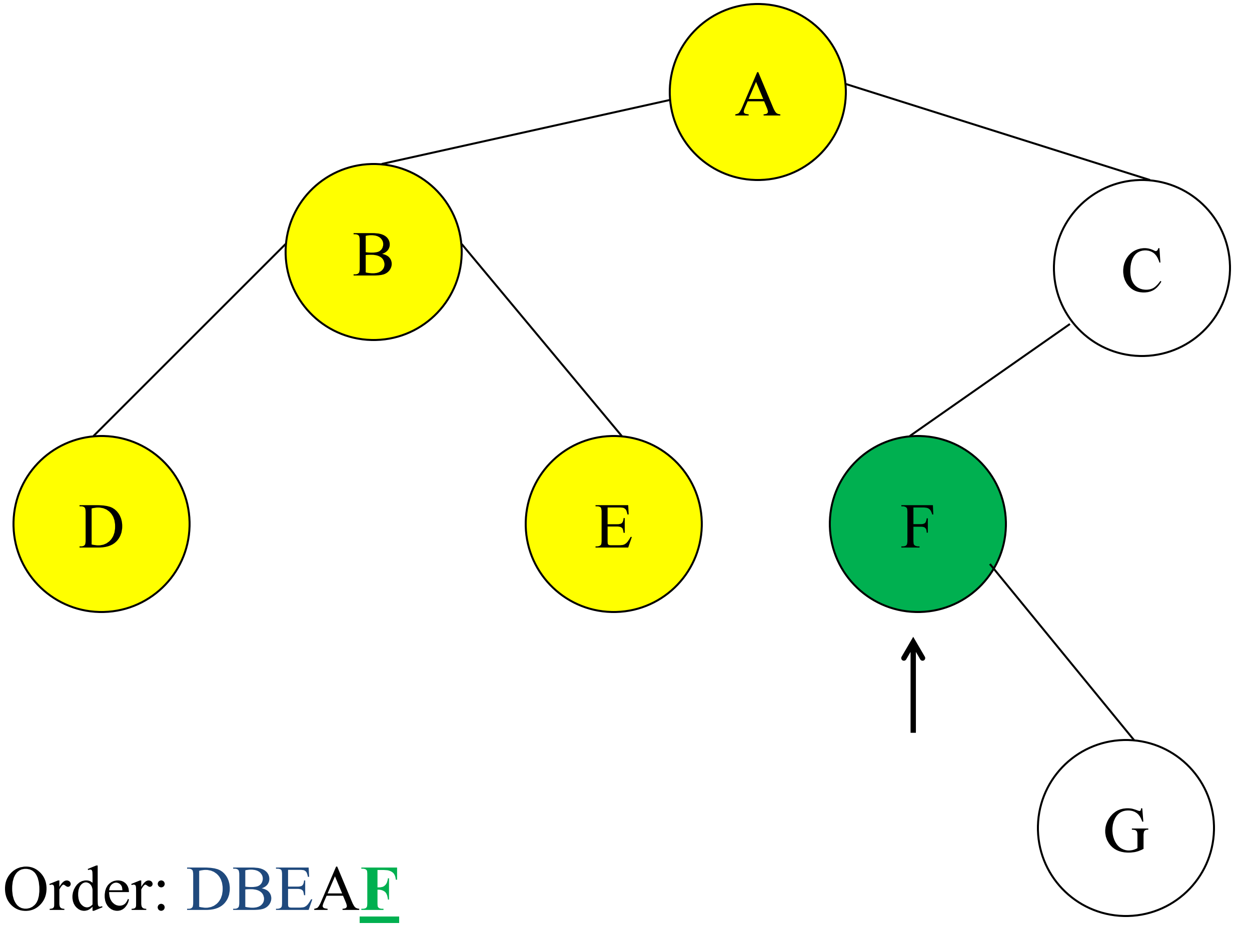

In Order: DBEA

# Tree Traversals: An Example



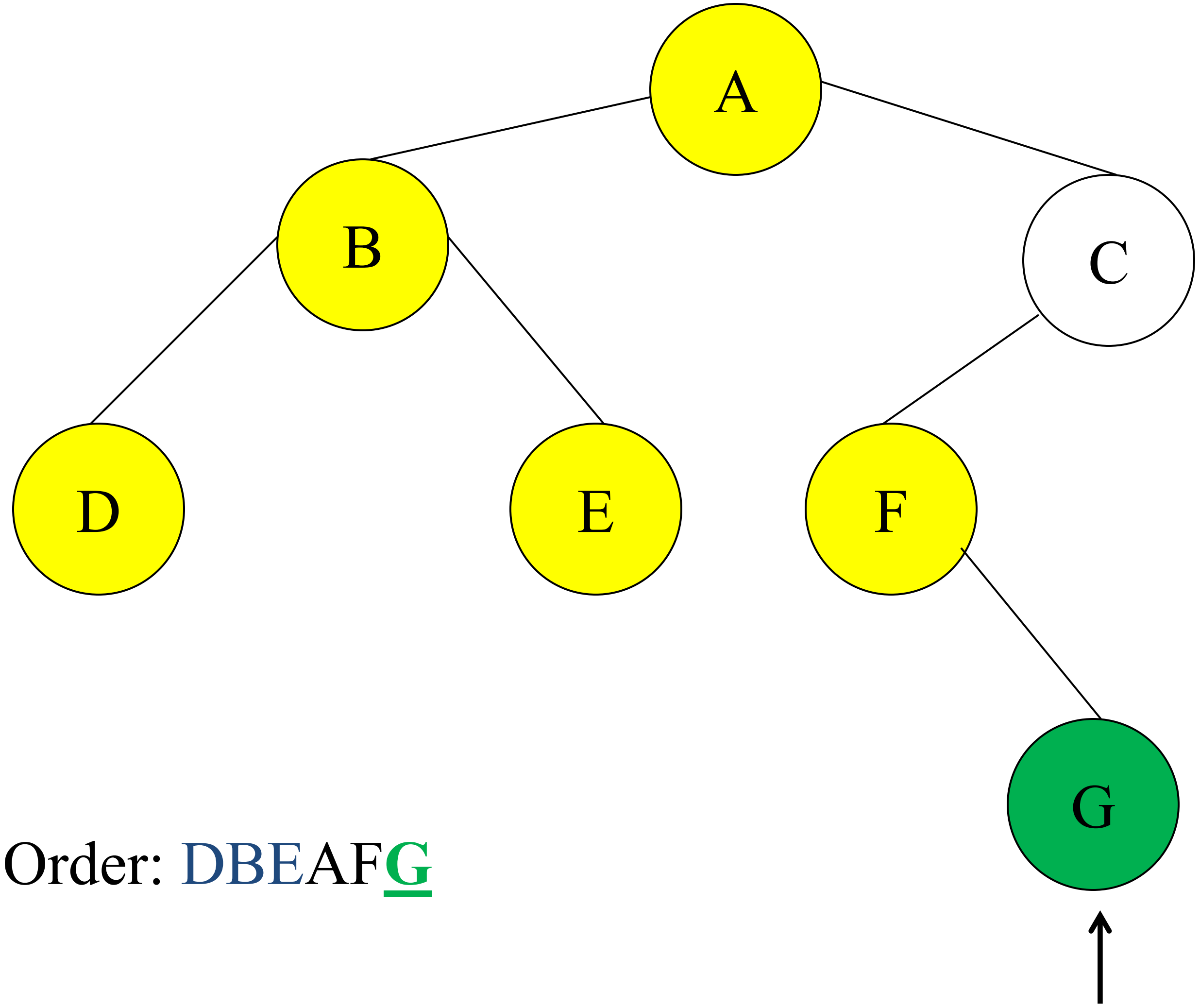

In Order: DBEA

# Tree Traversals: An Example



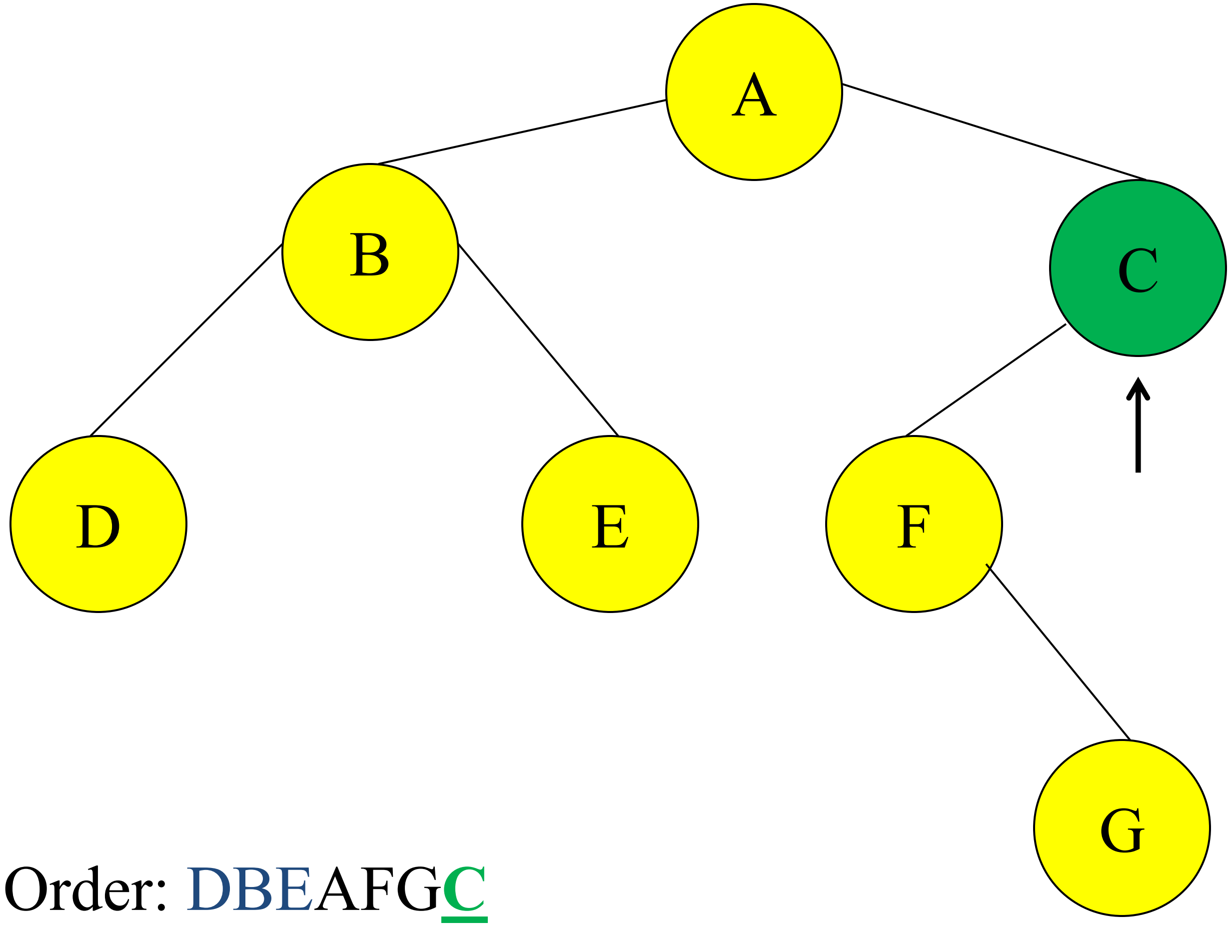

In Order: DBEA**F**

# Tree Traversals: An Example



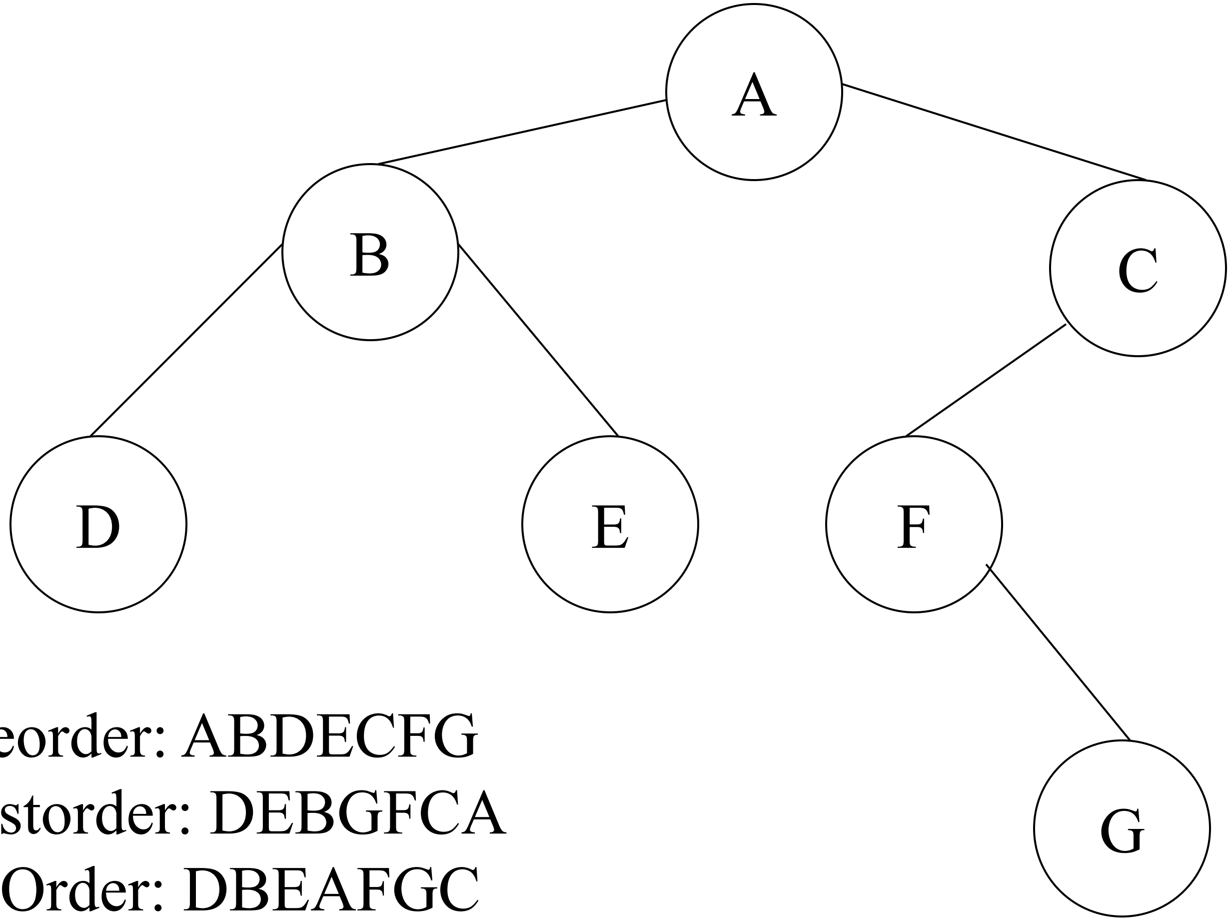

In Order: DBEAFG

# Tree Traversals: An Example



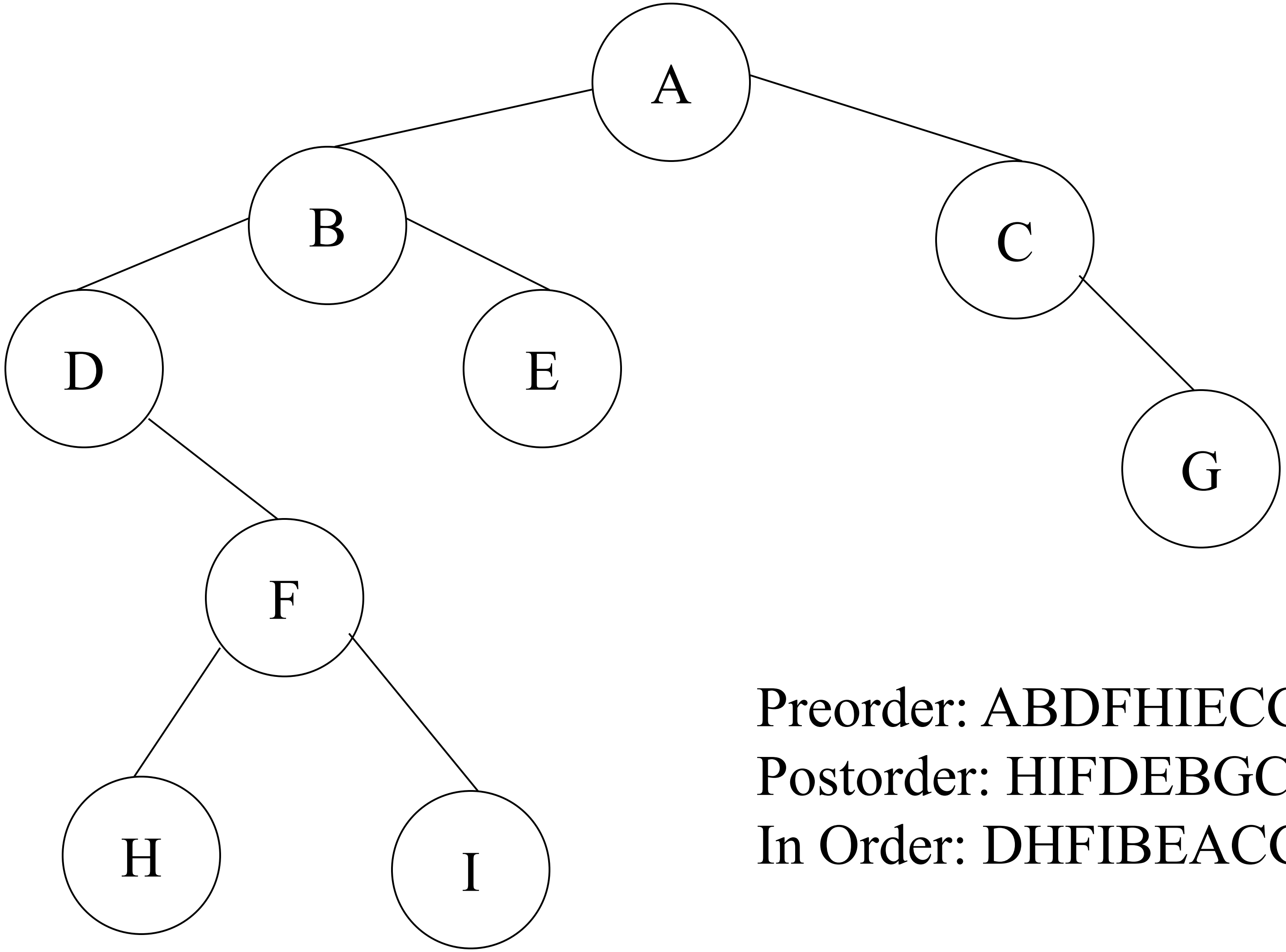

In Order: DBEAFG**C**

# Tree Traversals: An Example



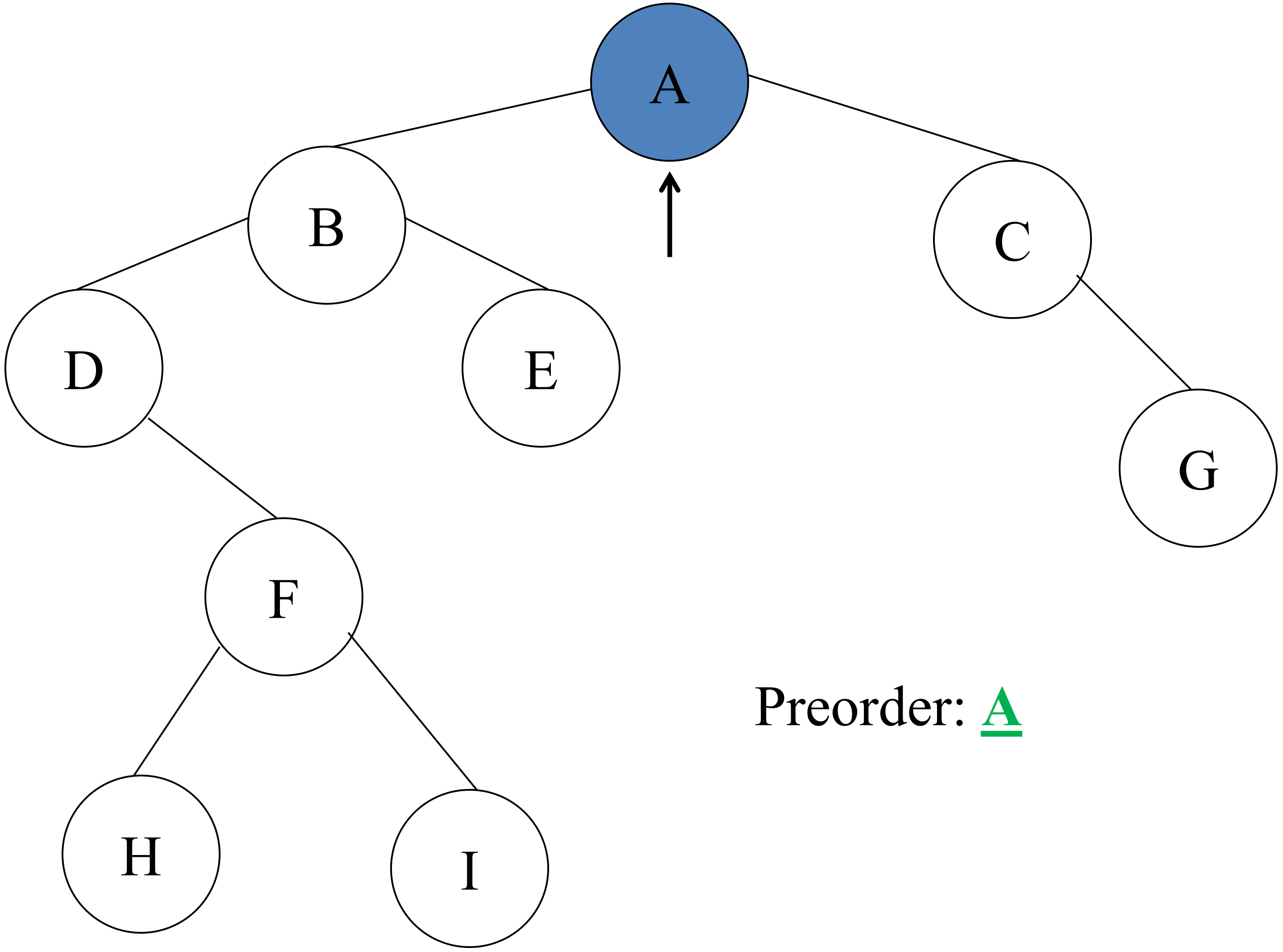

Preorder: ABDECFG
Postorder: DEBGFCA
In Order: DBEAFGC

# Tree Traversals: Another Example



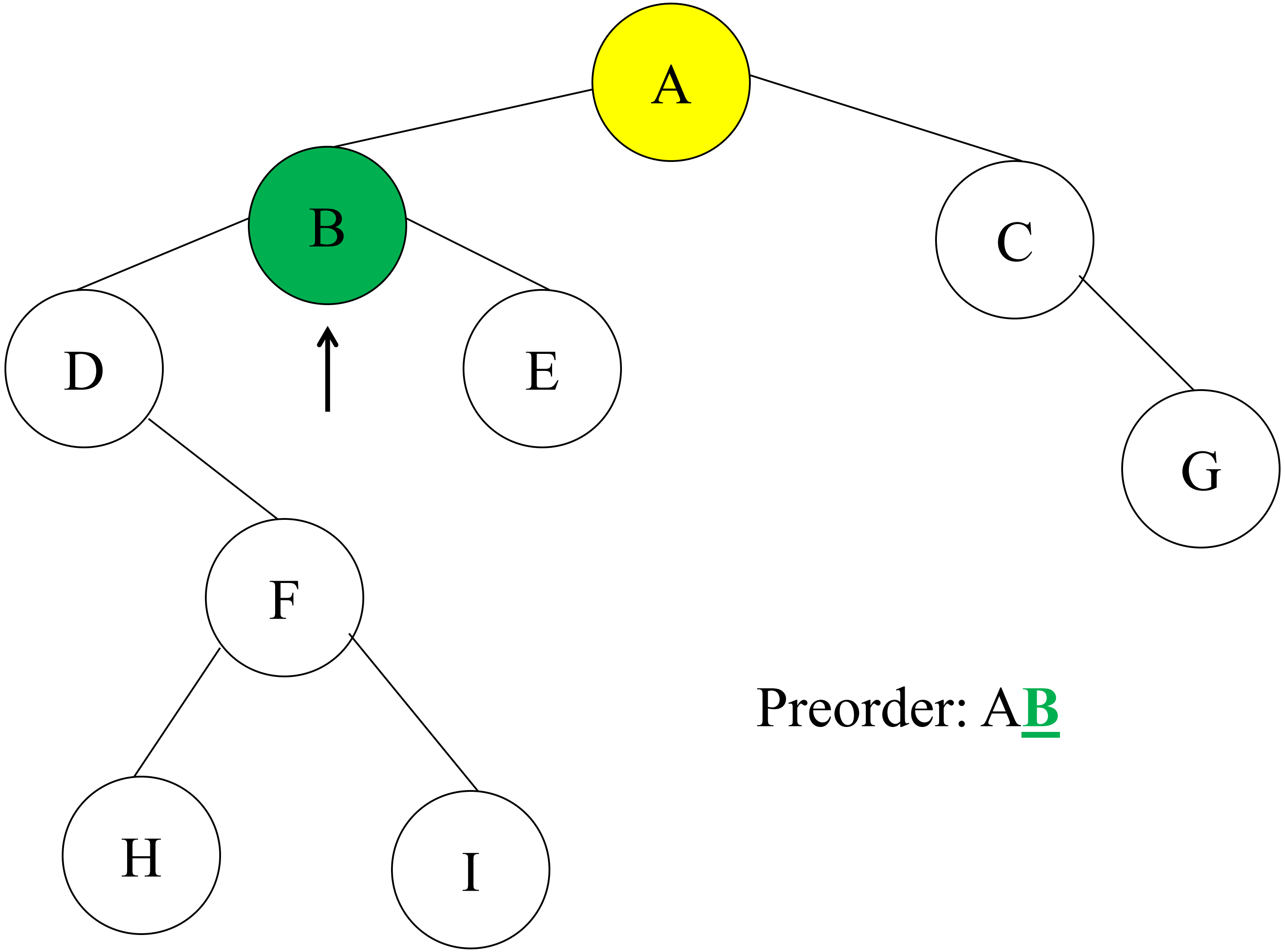

Preorder: ABDFHIECG
Postorder: HIFDEBGCA
In Order: DHFIBEACG

# Tree Traversals: Another Example



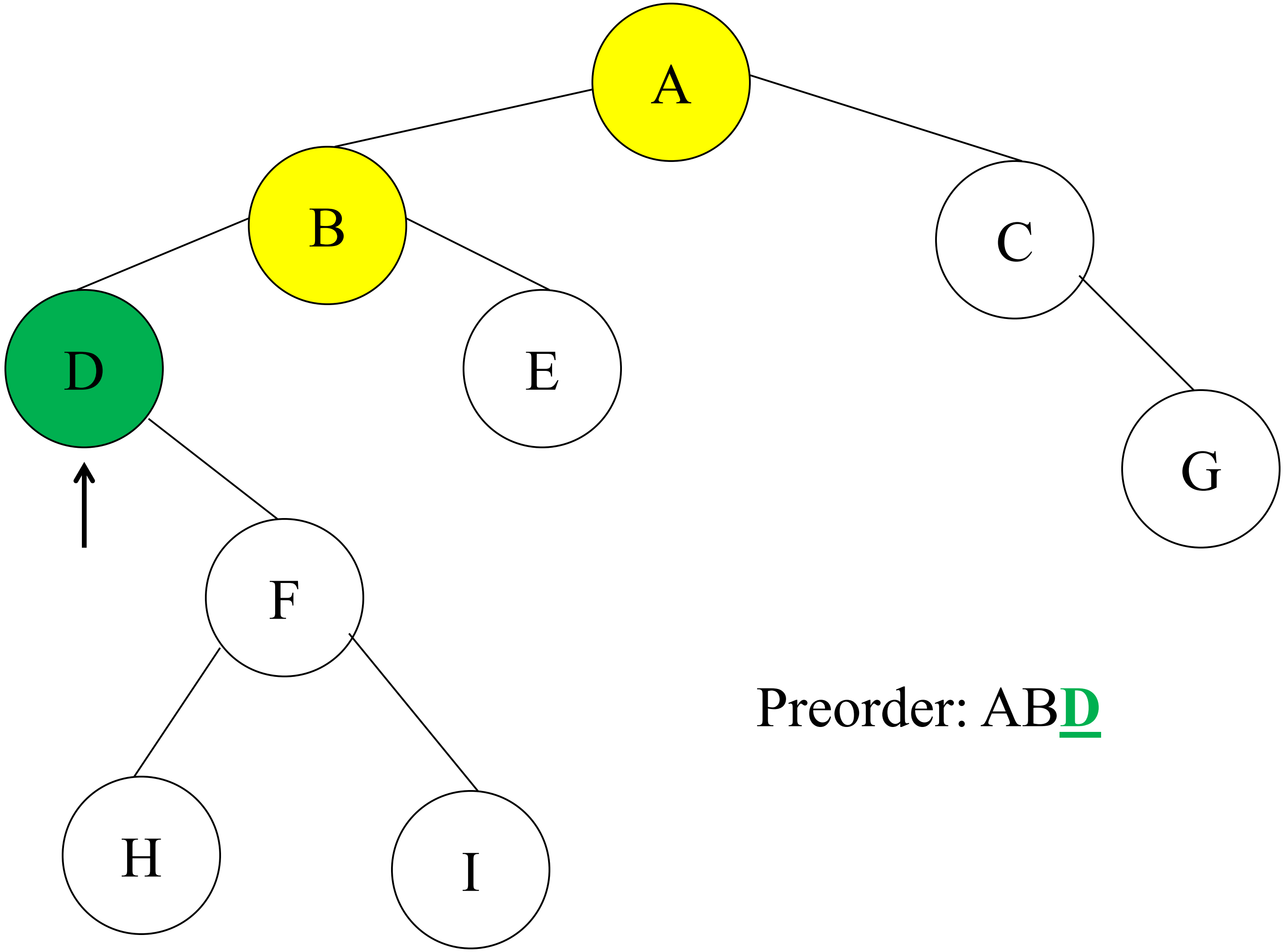

Preorder: **A**

# Tree Traversals: Another Example



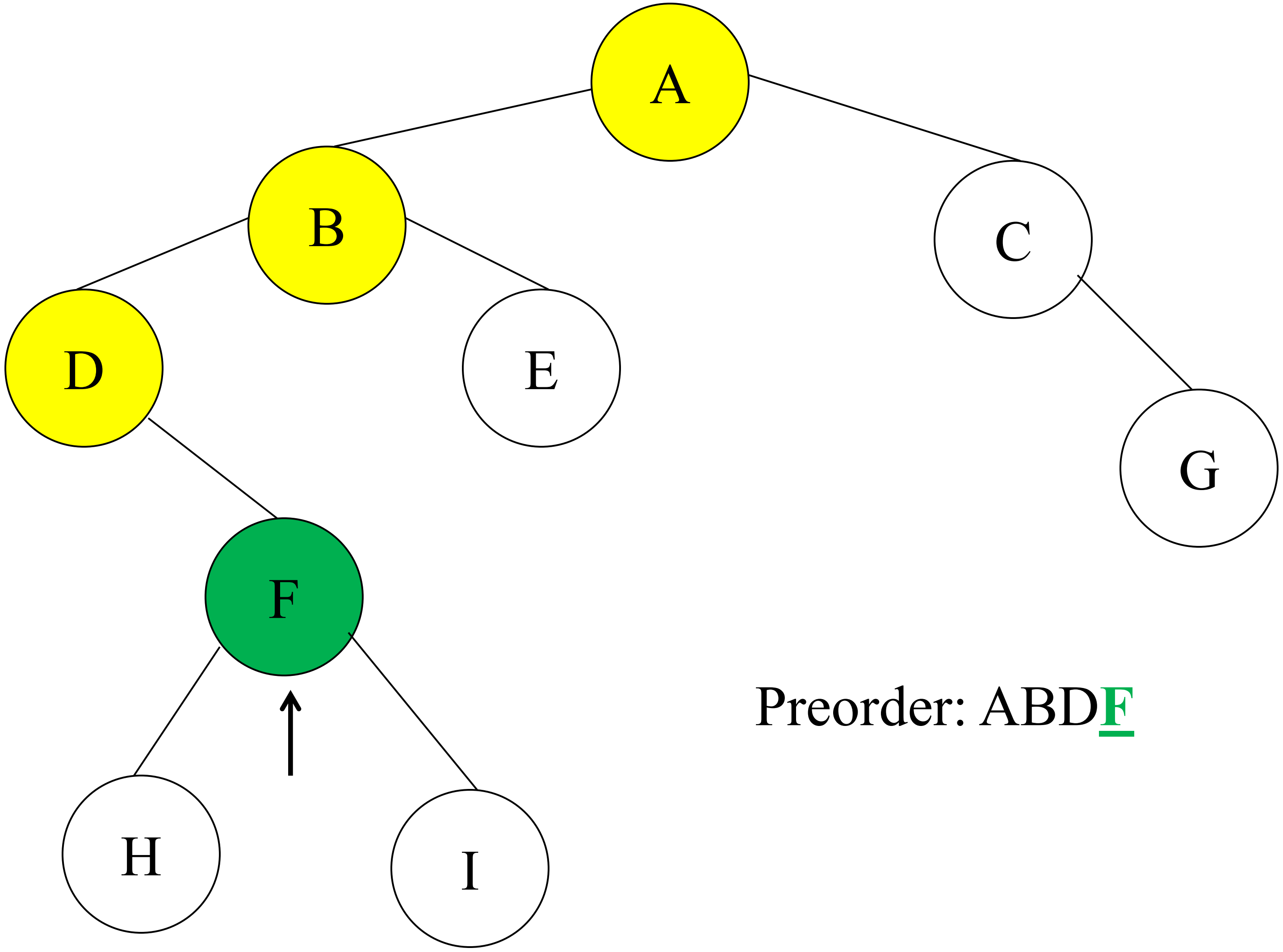

Preorder: A**B**

# Tree Traversals: Another Example



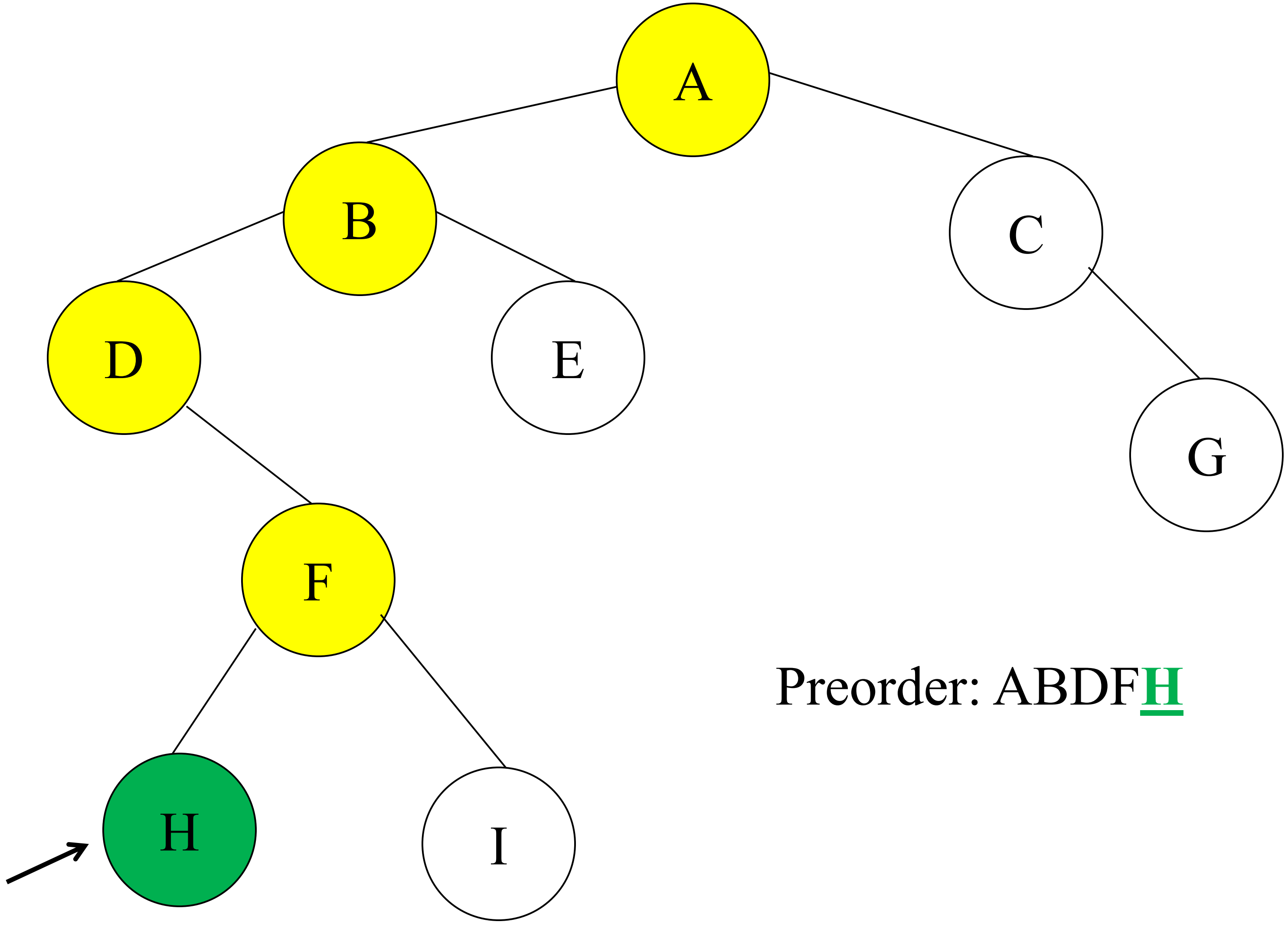

Preorder: AB**<u>D</u>**

# Tree Traversals: Another Example



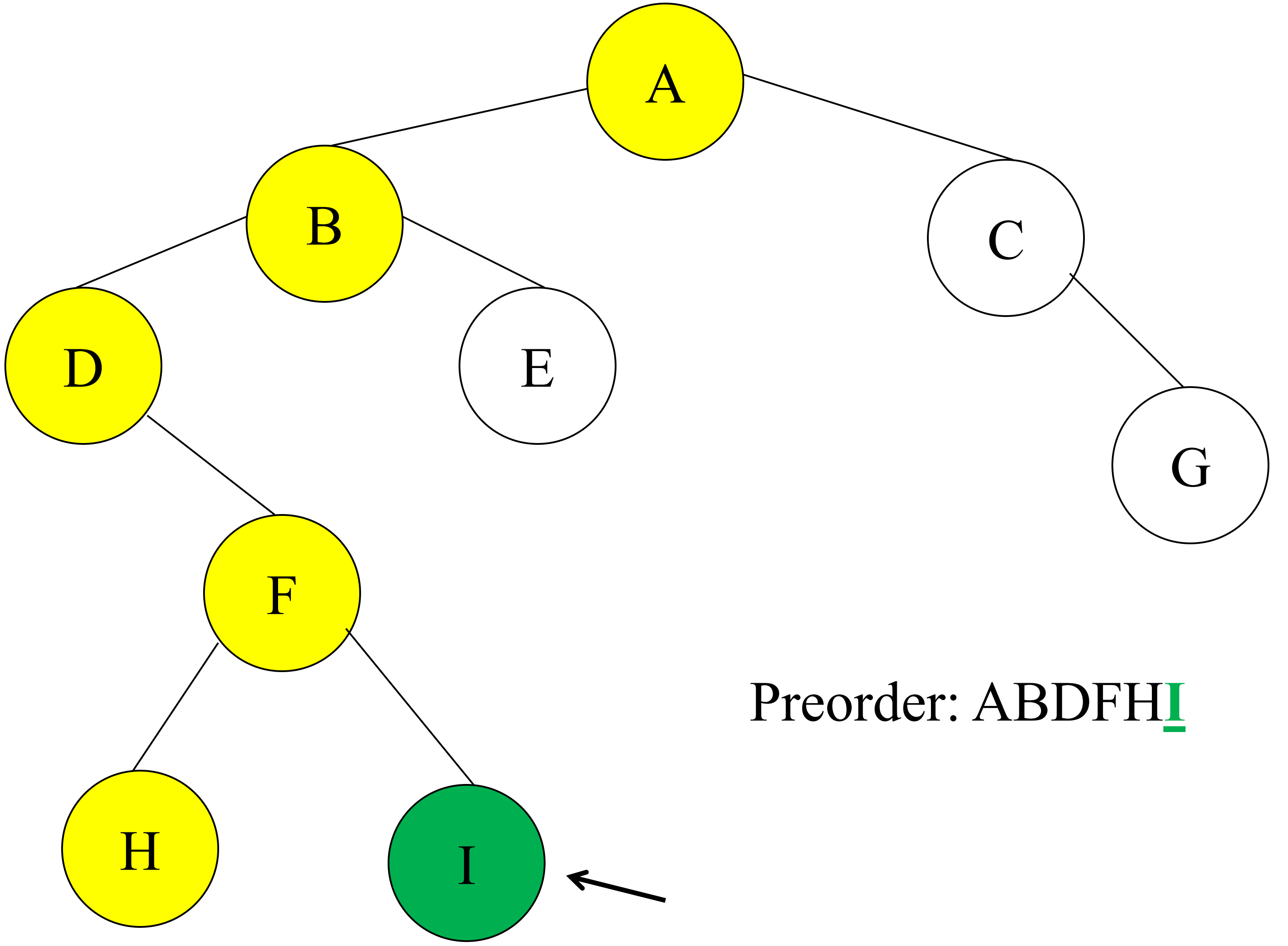

Preorder: ABD**F**

# Tree Traversals: Another Example



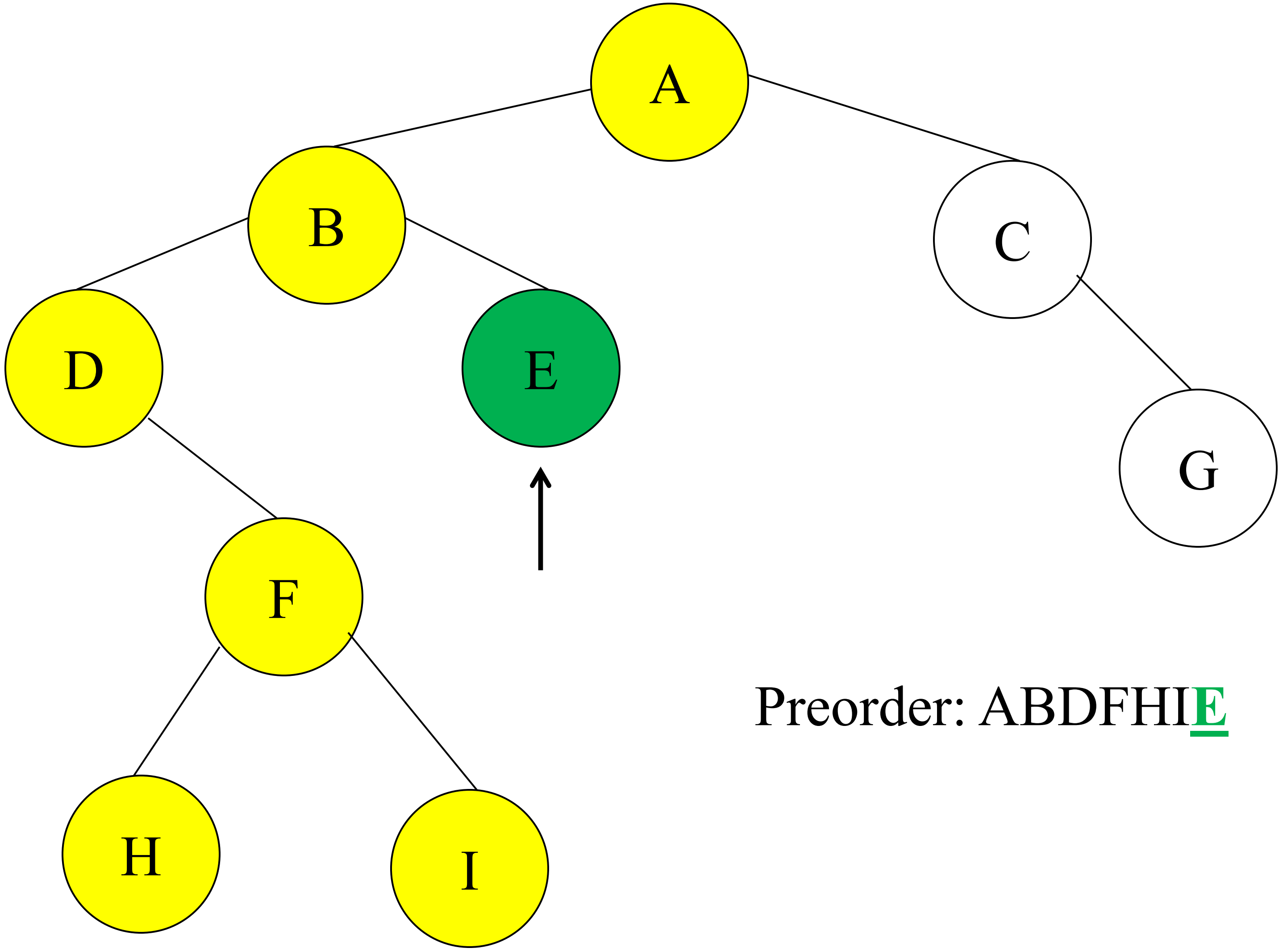

Preorder: ABDF**H**

# Tree Traversals: Another Example



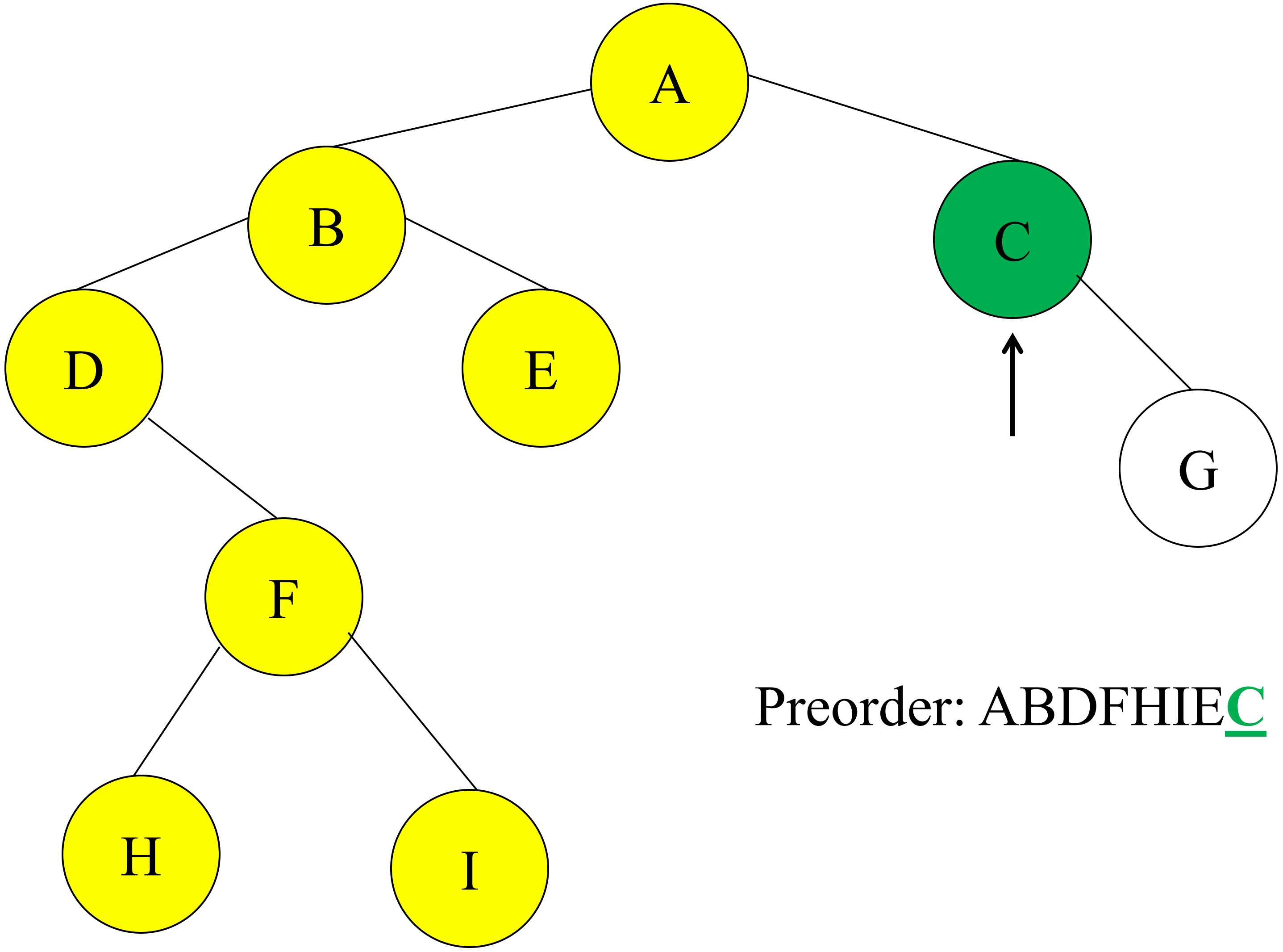

Preorder: ABDFHI

# Tree Traversals: Another Example



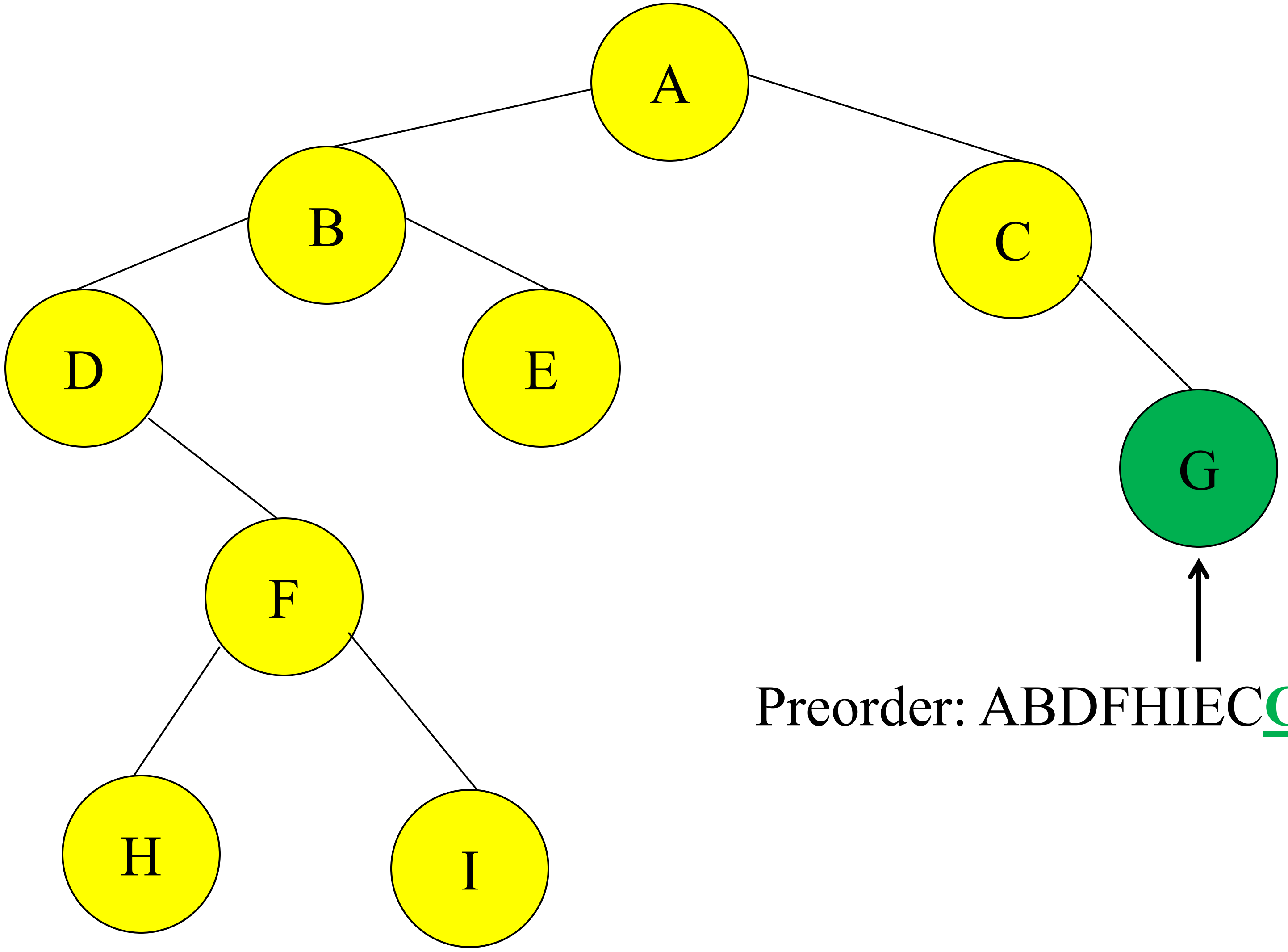

Preorder: ABDFHI**E**

# Tree Traversals: Another Example



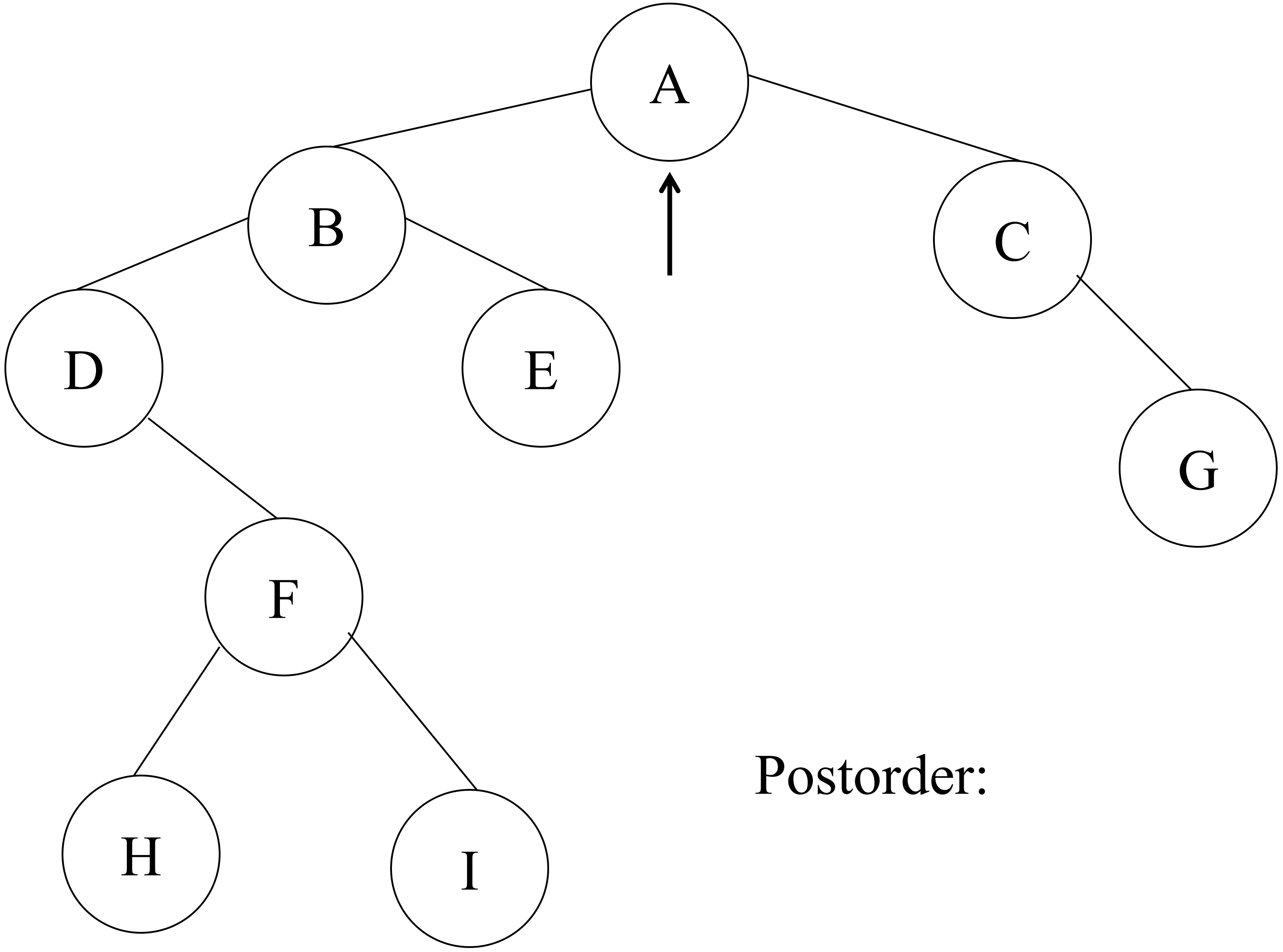

Preorder: ABDFHIE**C**

# Tree Traversals: Another Example



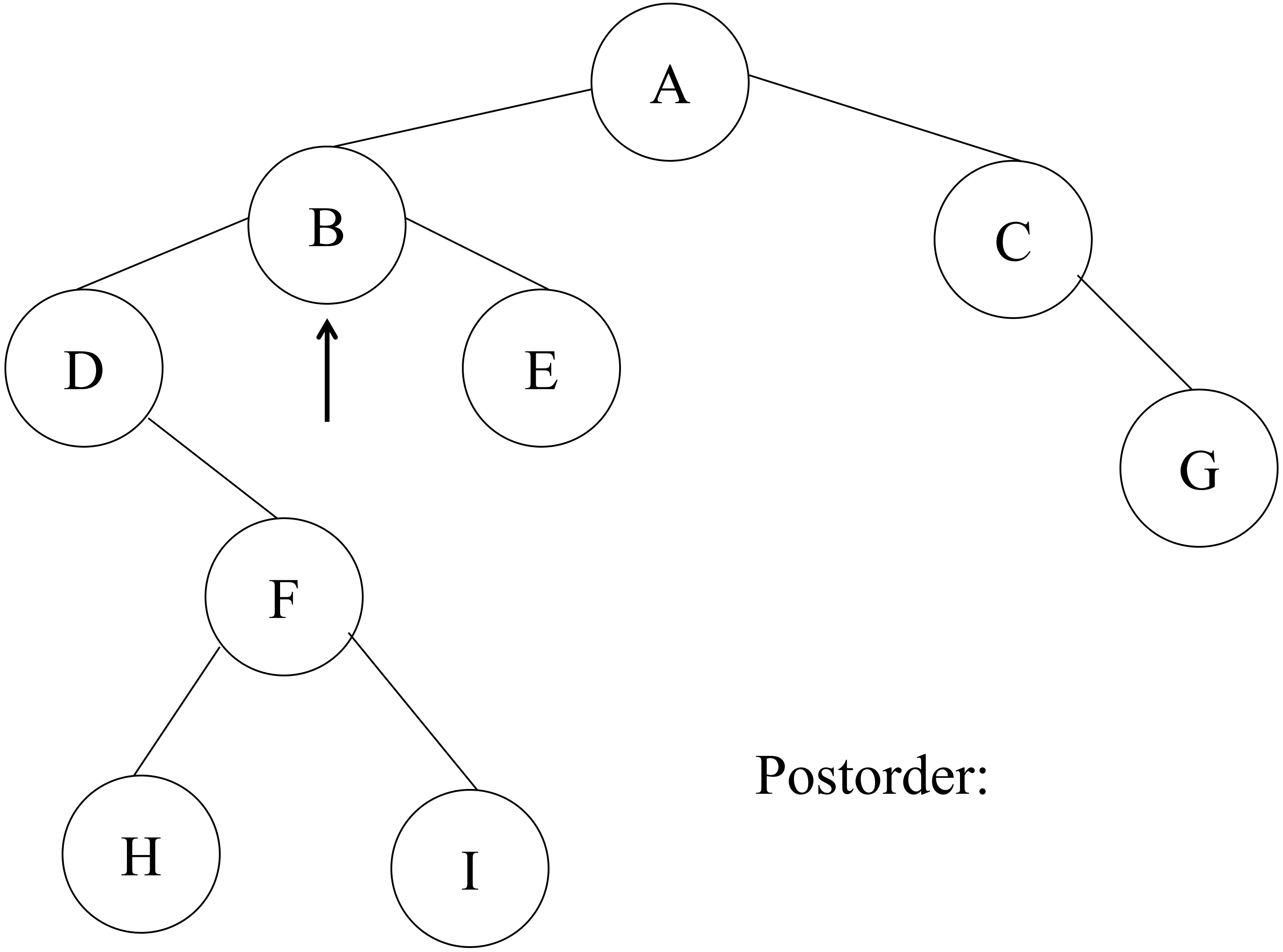

Preorder: ABDFHIEC**G**

# Tree Traversals: Another Example



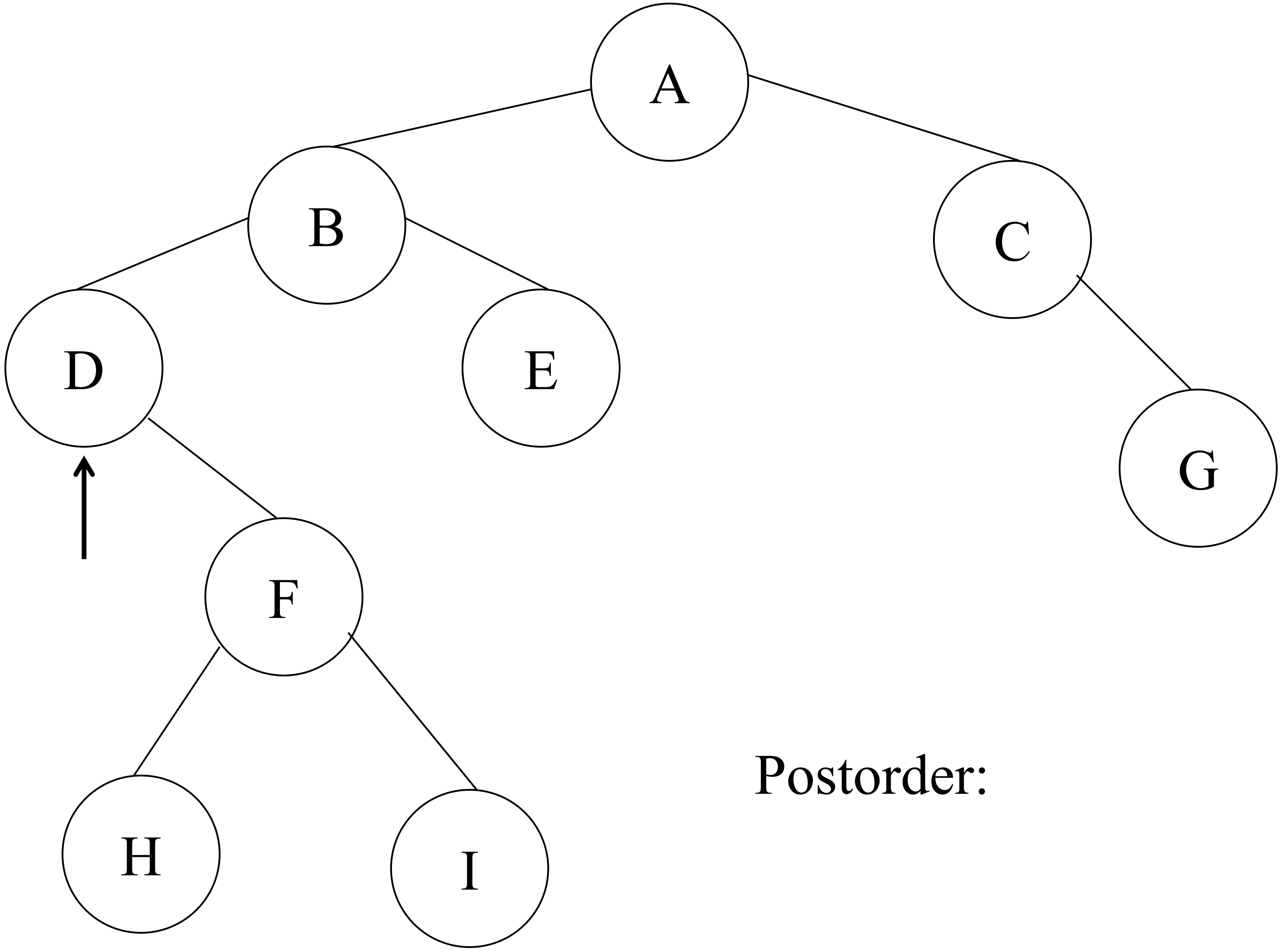

Postorder:

# Tree Traversals: Another Example



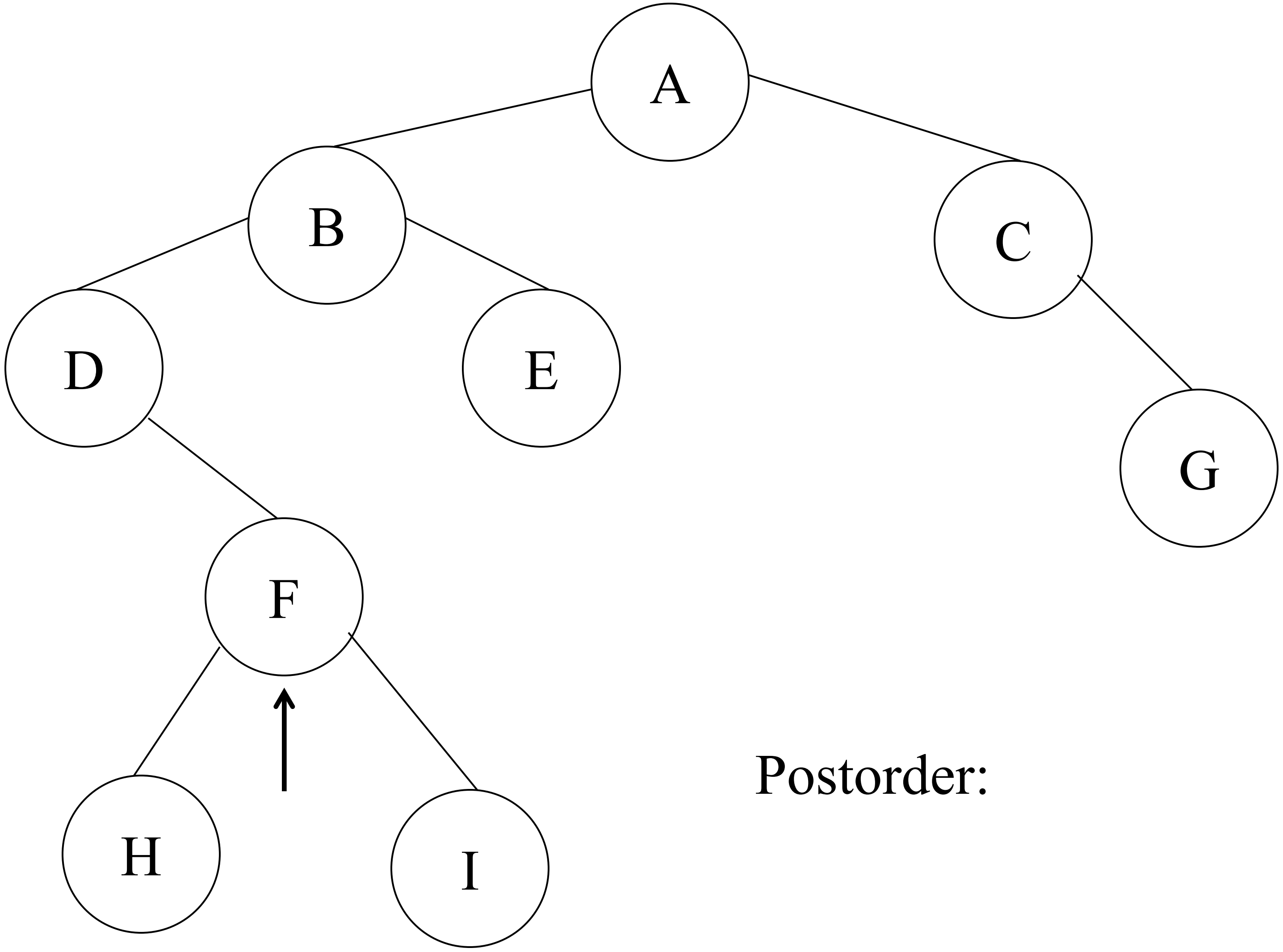

Postorder:

# Tree Traversals: Another Example



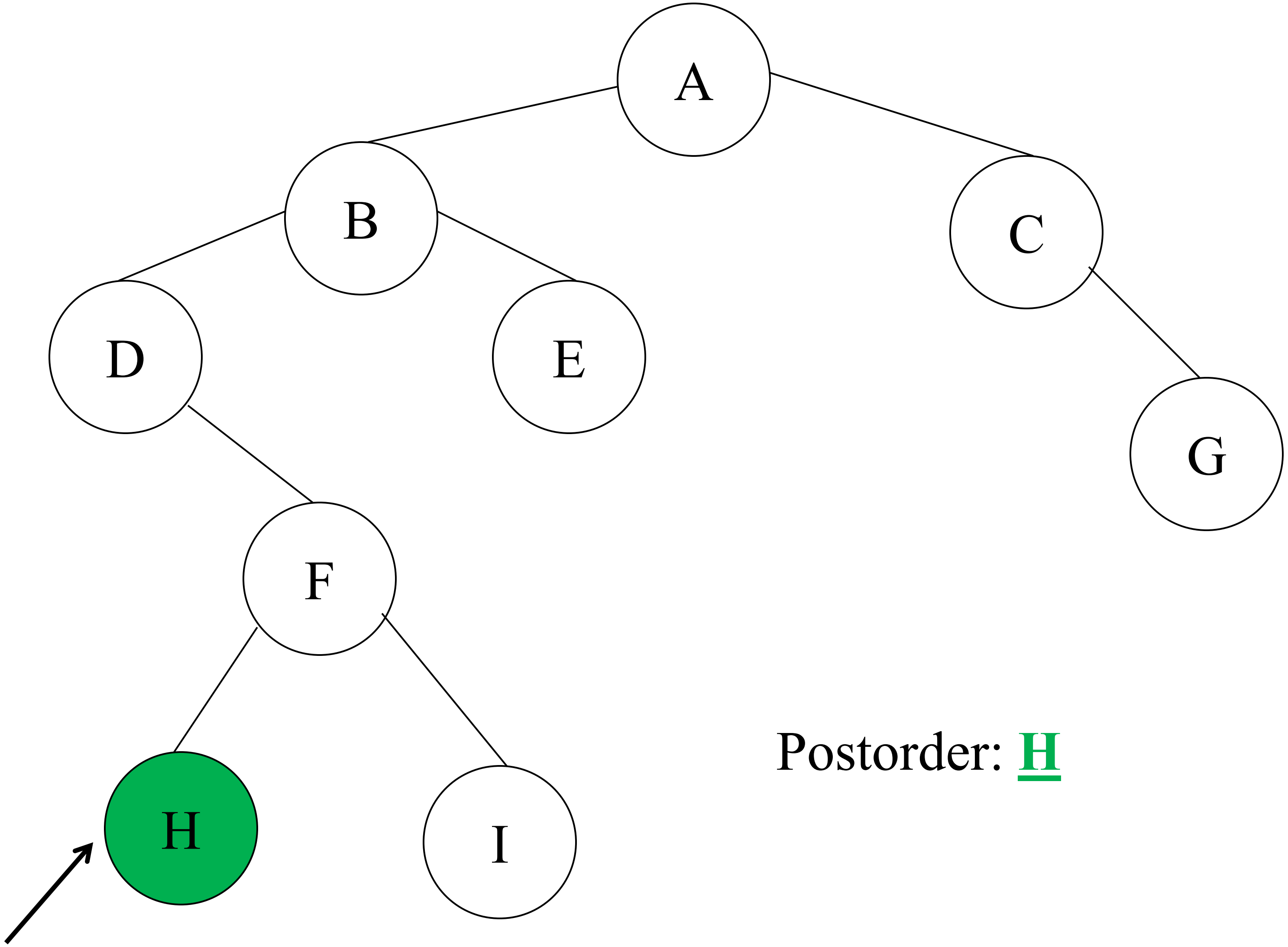

Postorder:

# Tree Traversals: Another Example



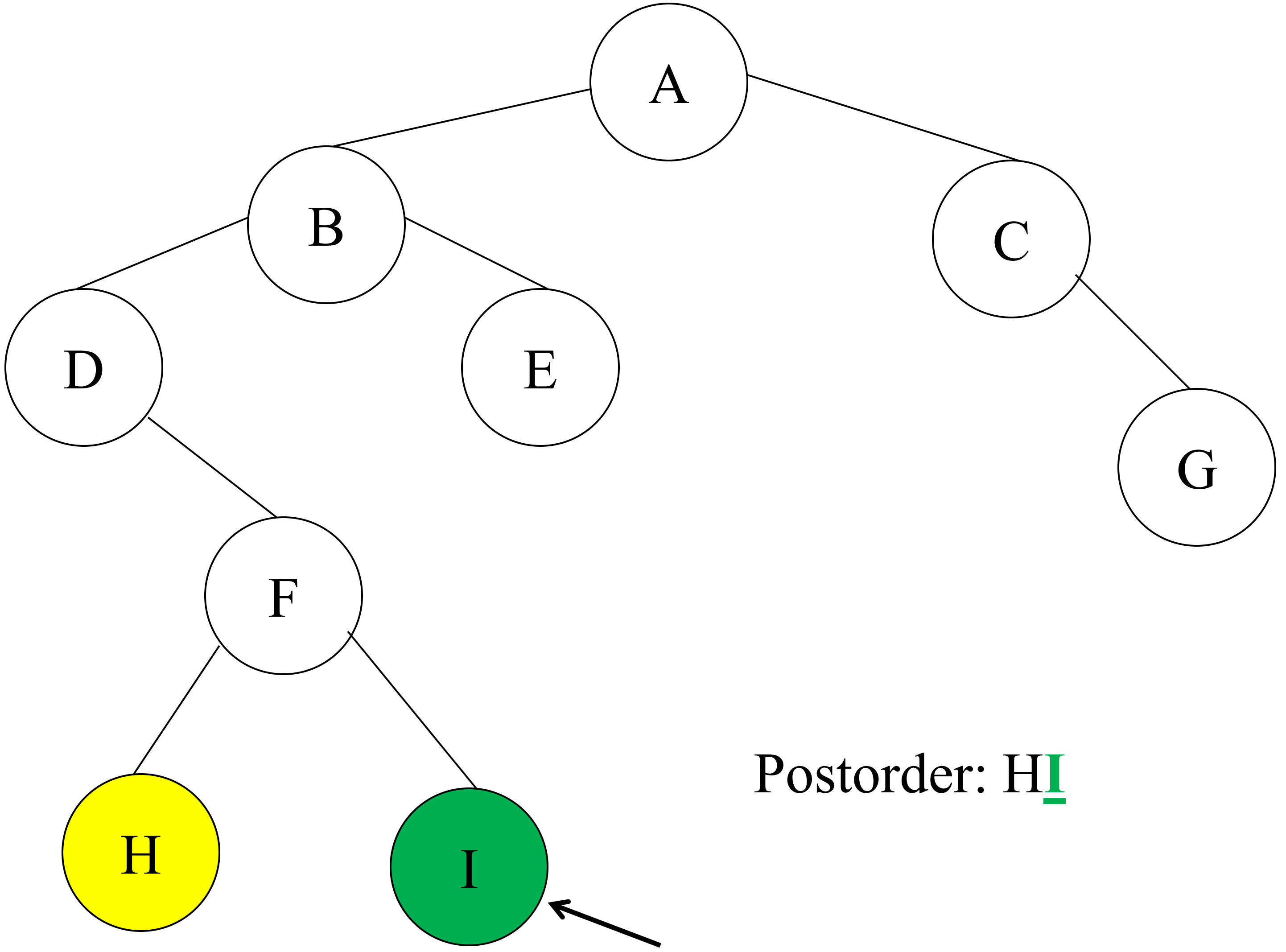

Postorder:

# Tree Traversals: Another Example



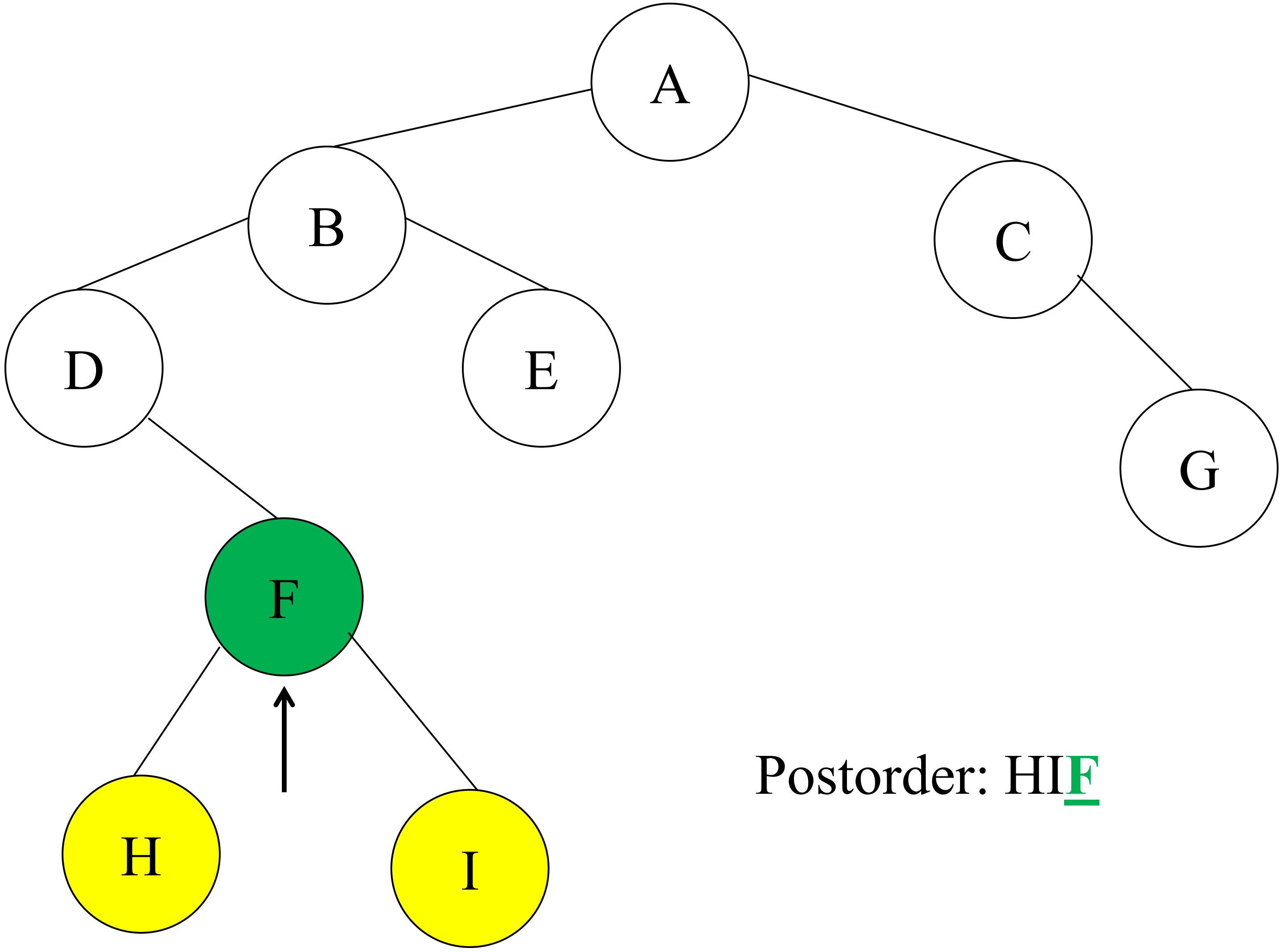

Postorder: **H**

# Tree Traversals: Another Example



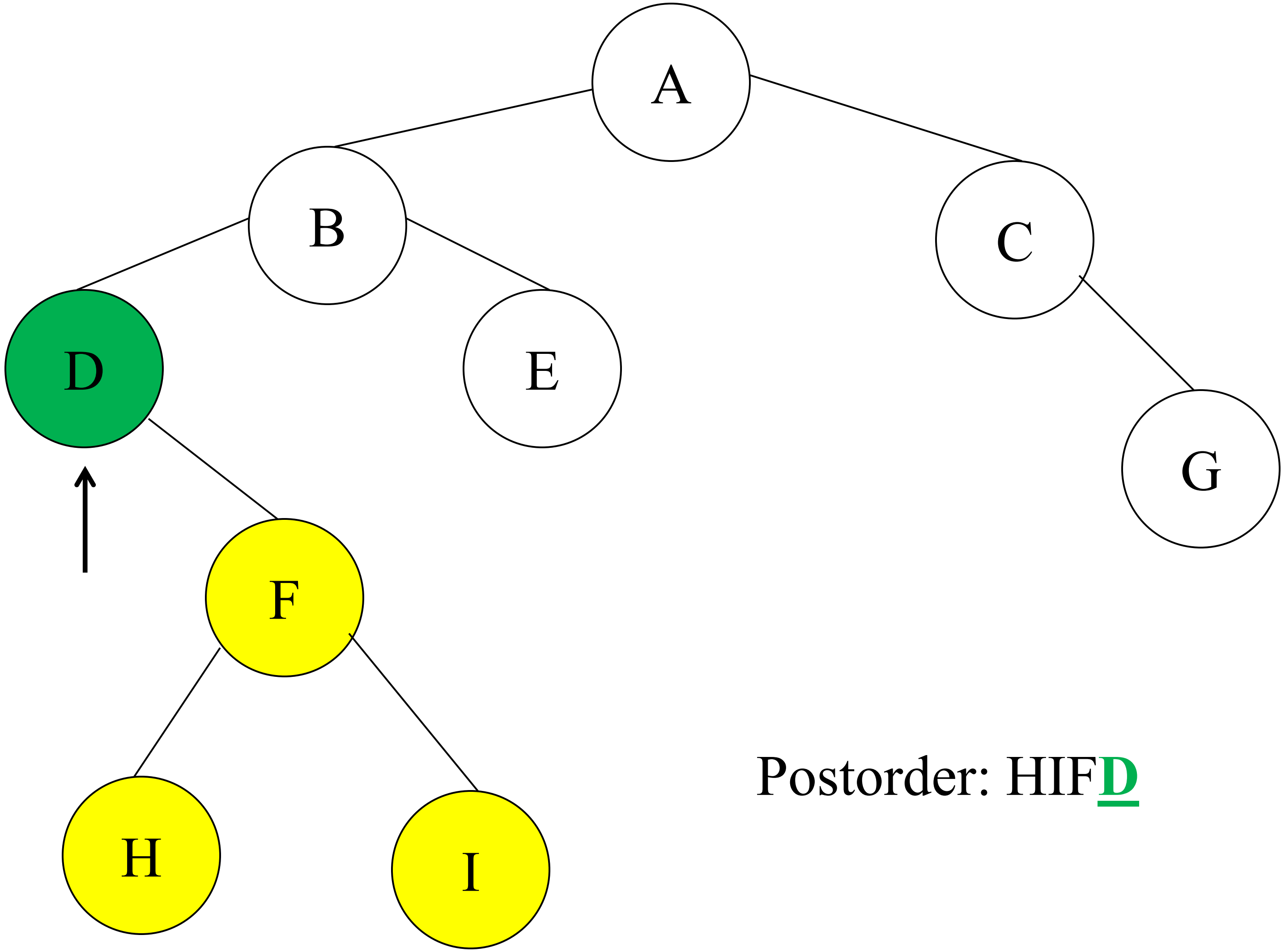

Postorder: H**I**

# Tree Traversals: Another Example



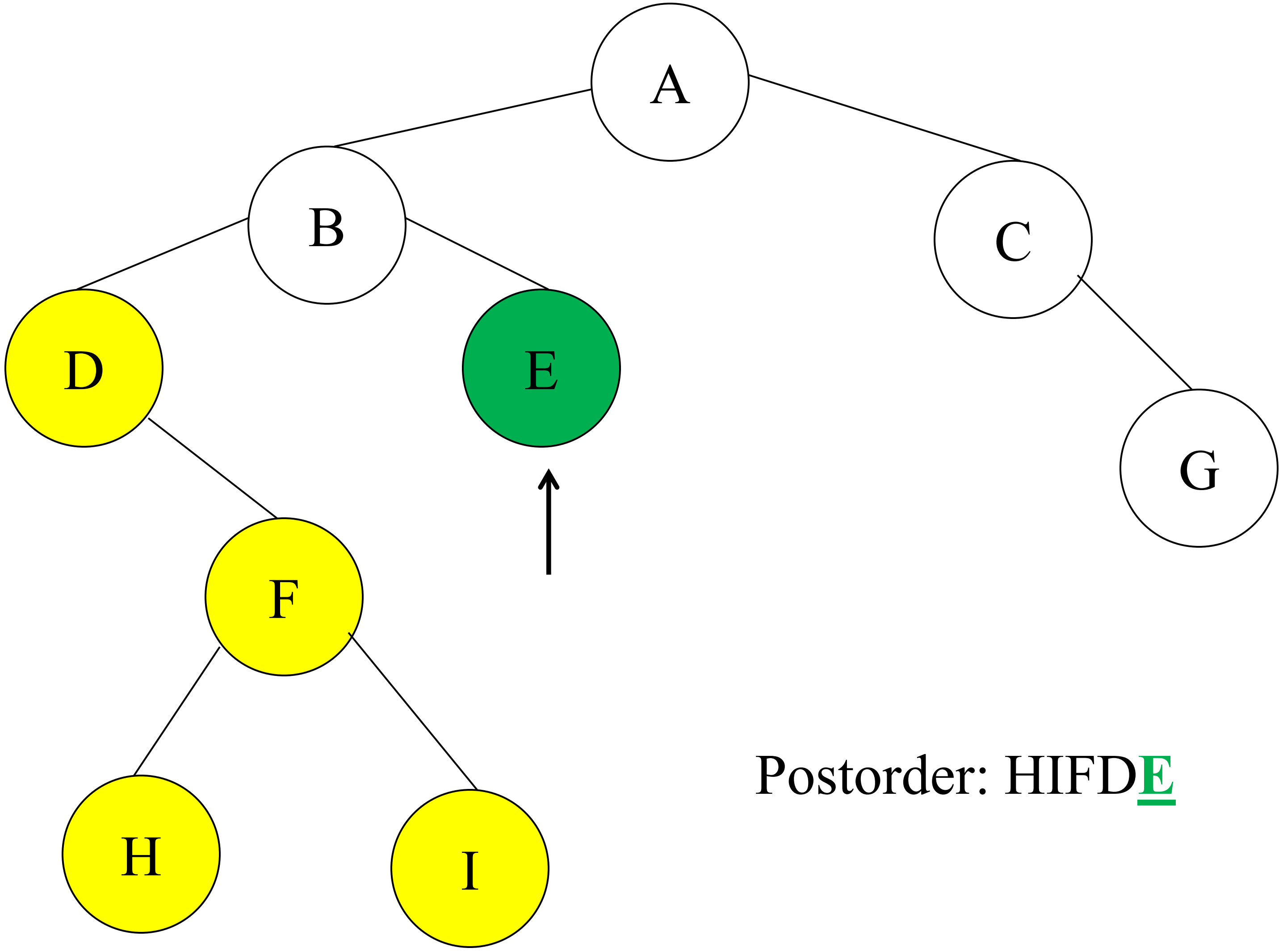

Postorder: HI**F**

# Tree Traversals: Another Example



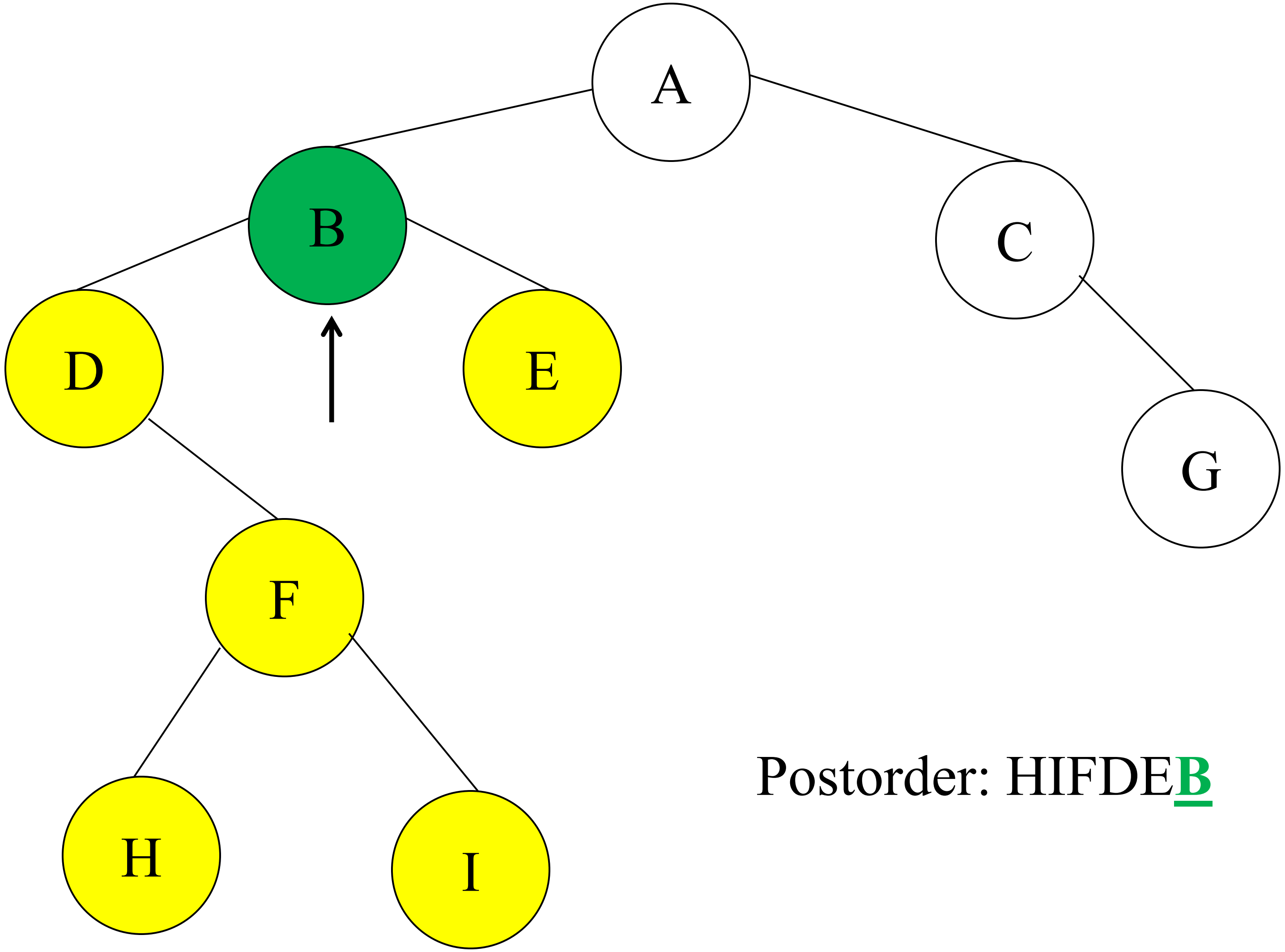

Postorder: HIF**D**

# Tree Traversals: Another Example



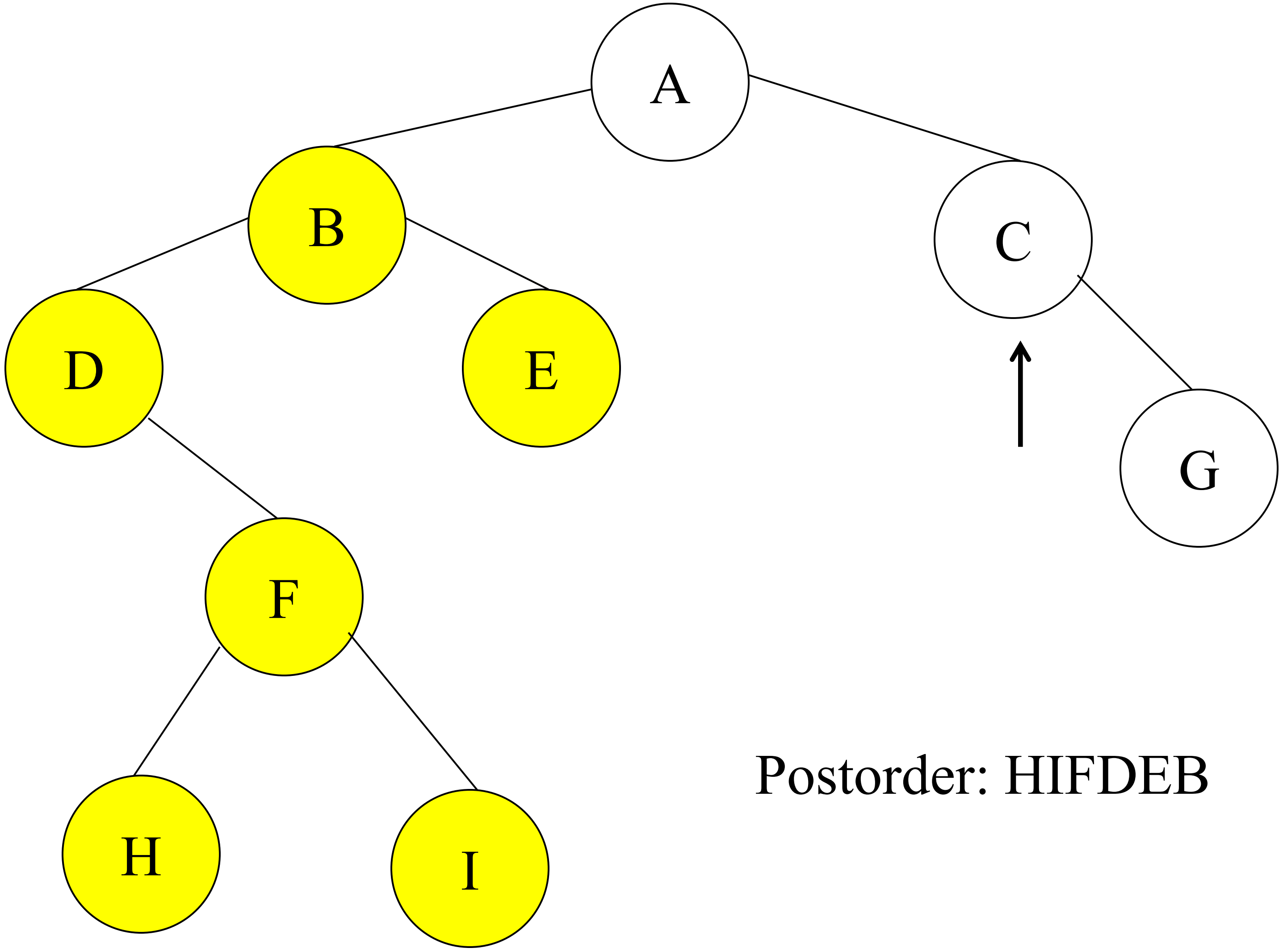

Postorder: HIFD**E**

# Tree Traversals: Another Example



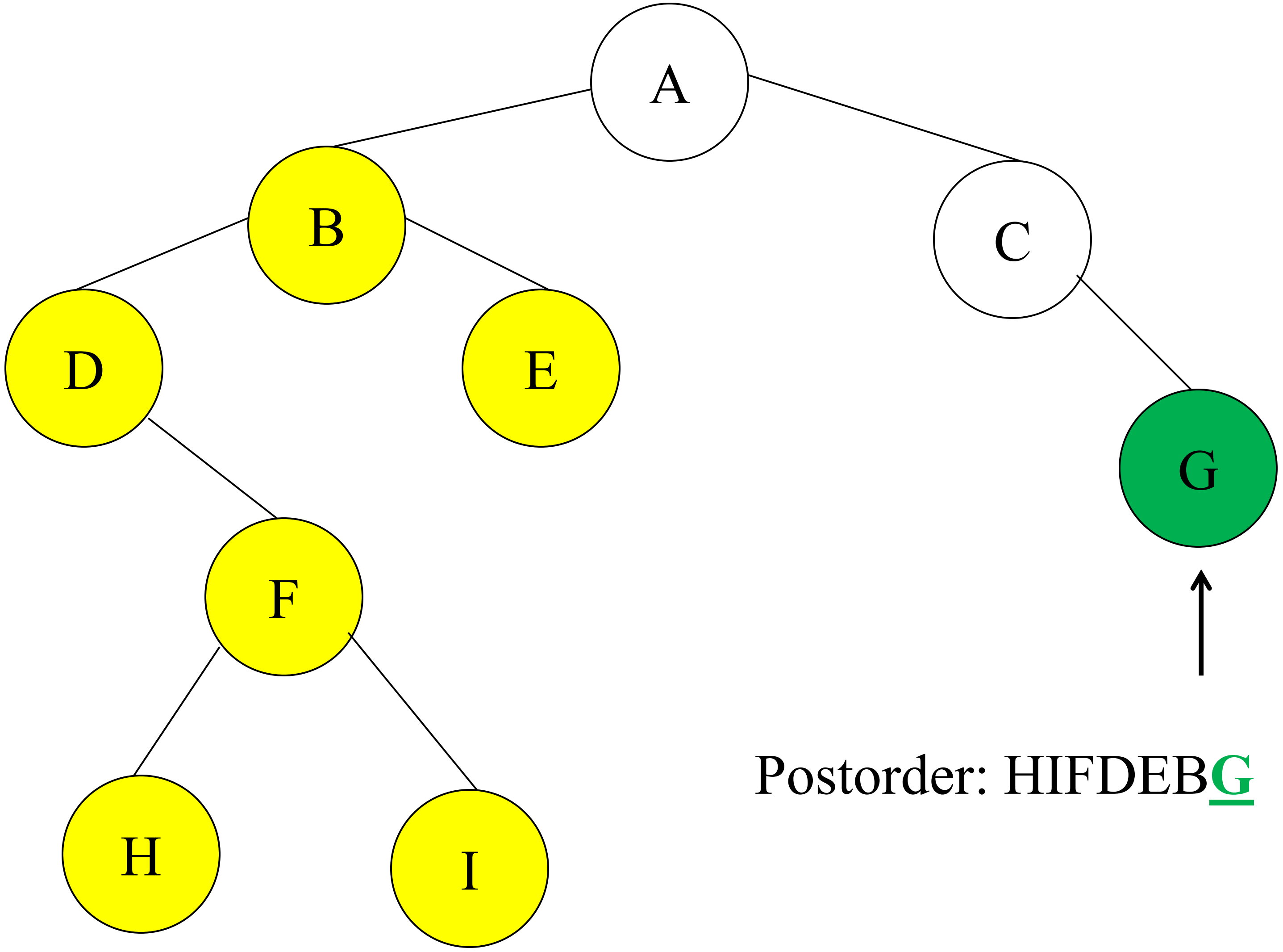

Postorder: HIFDE**B**

# Tree Traversals: Another Example



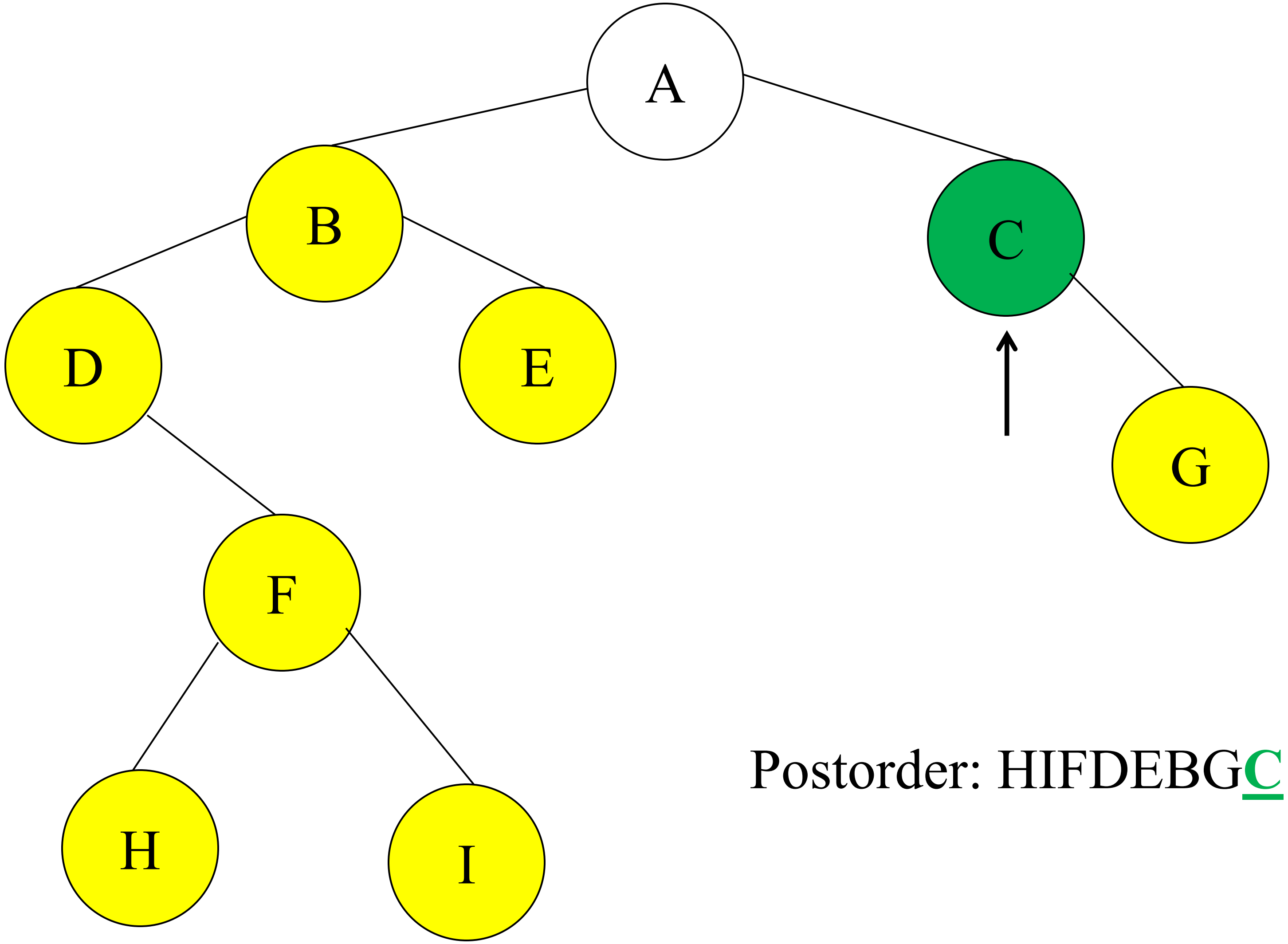

Postorder: HIFDEB

# Tree Traversals: Another Example



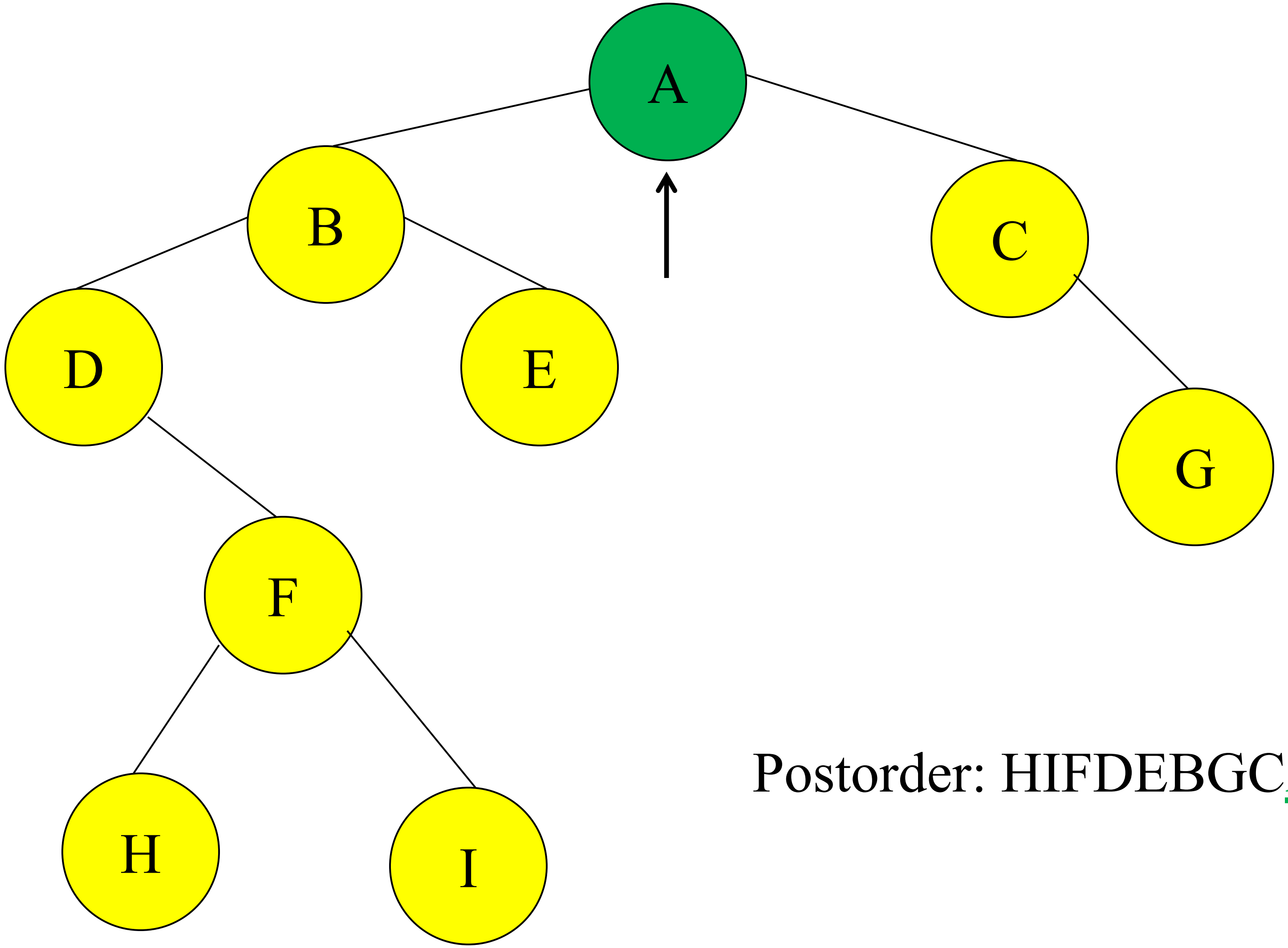

Postorder: HIFDEB**G**

# Tree Traversals: Another Example



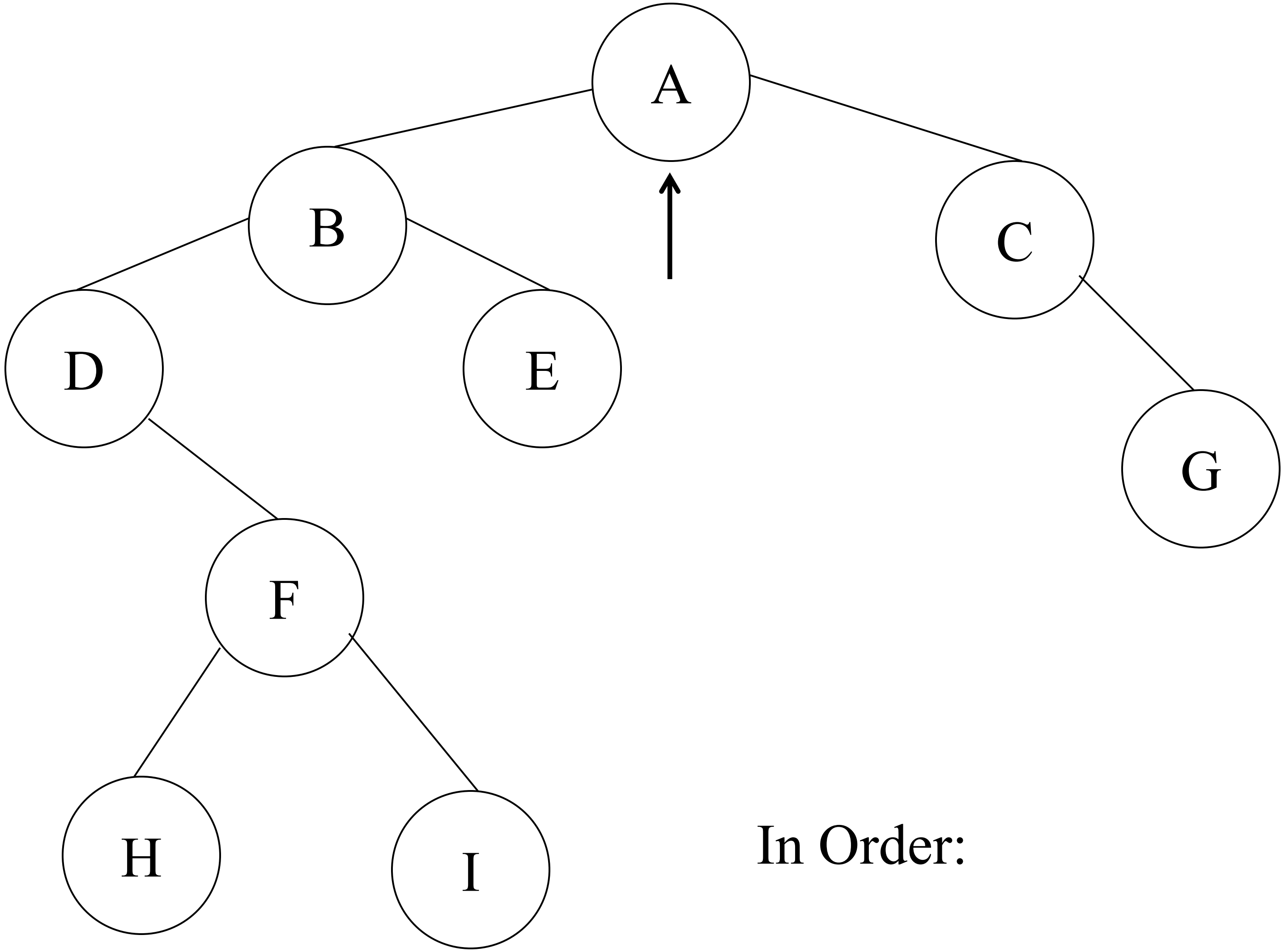

Postorder: HIFDEBG**C**

# Tree Traversals: Another Example



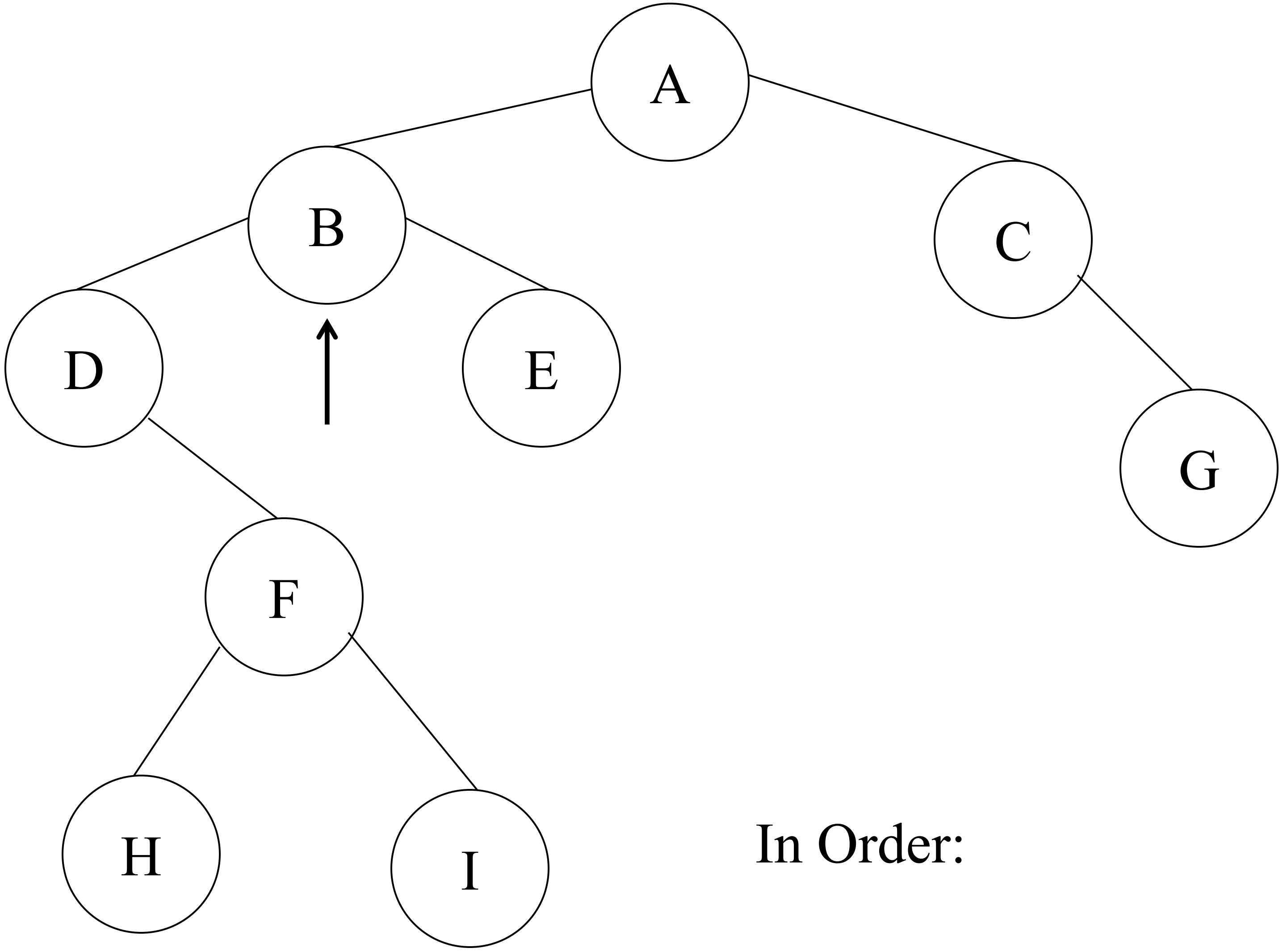

Postorder: HIFDEBGC**A**

# Tree Traversals: Another Example



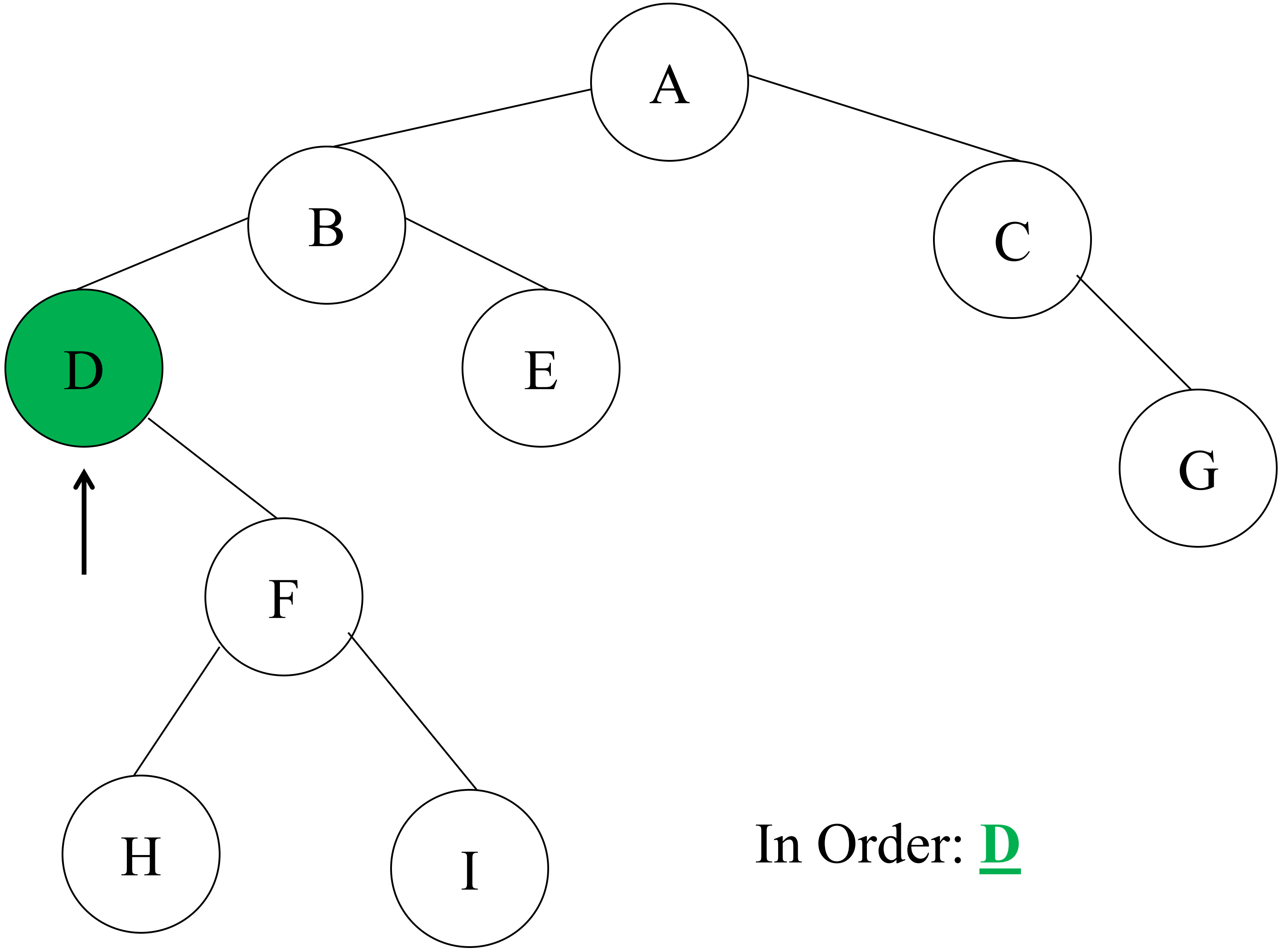

In Order:

# Tree Traversals: Another Example



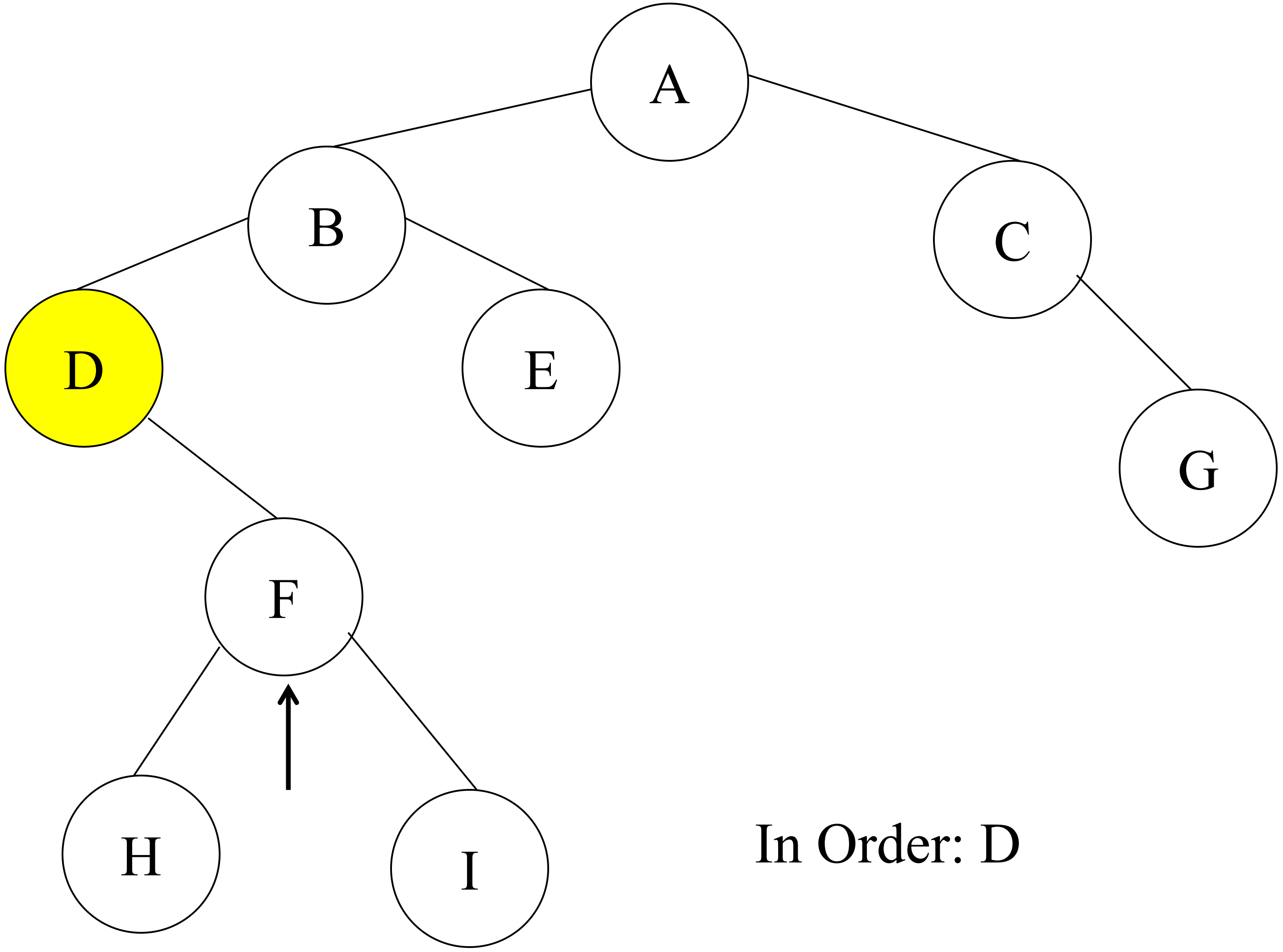

In Order:

# Tree Traversals: Another Example



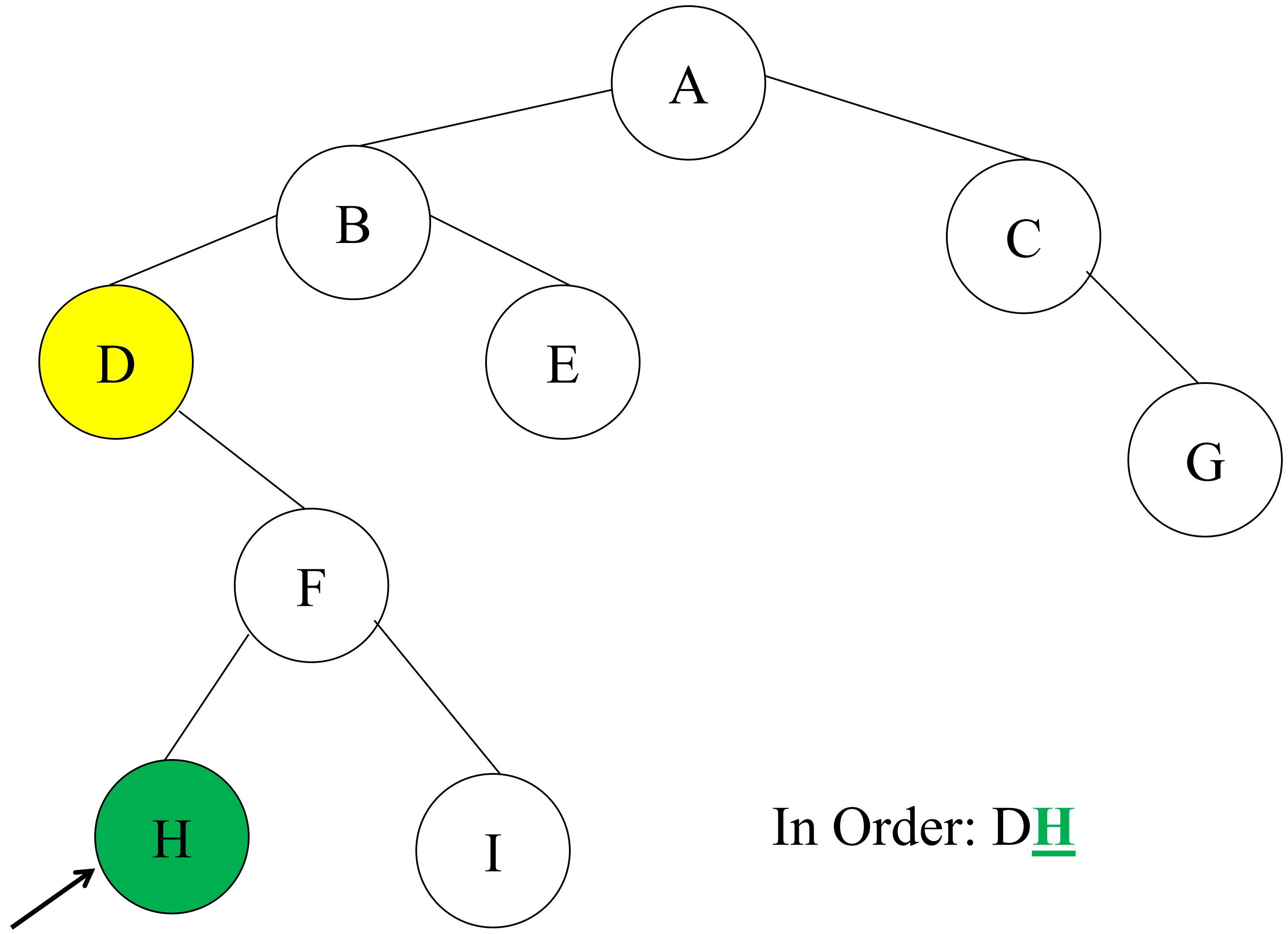

In Order: **D**

# Tree Traversals: Another Example



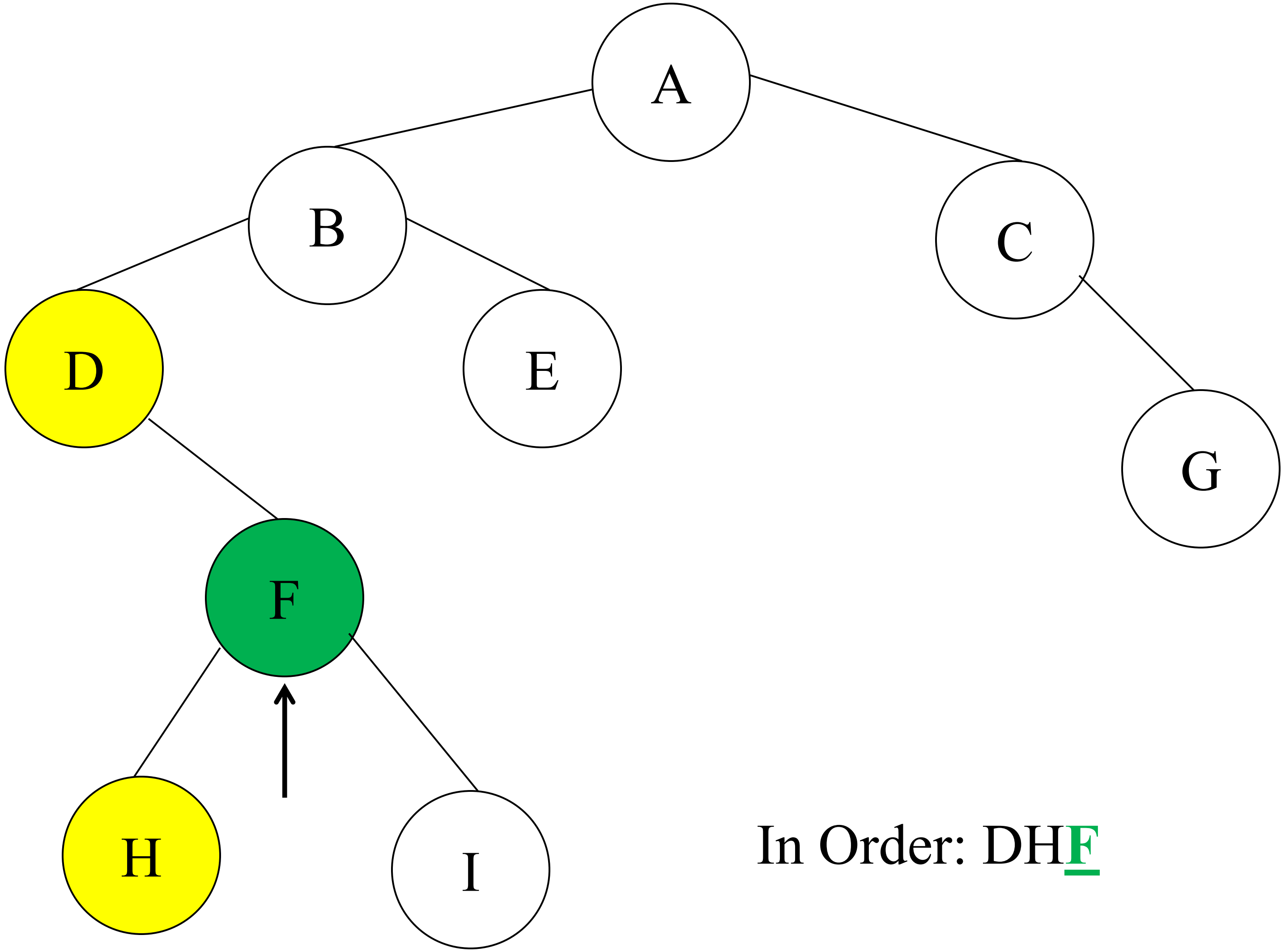

In Order: D

# Tree Traversals: Another Example



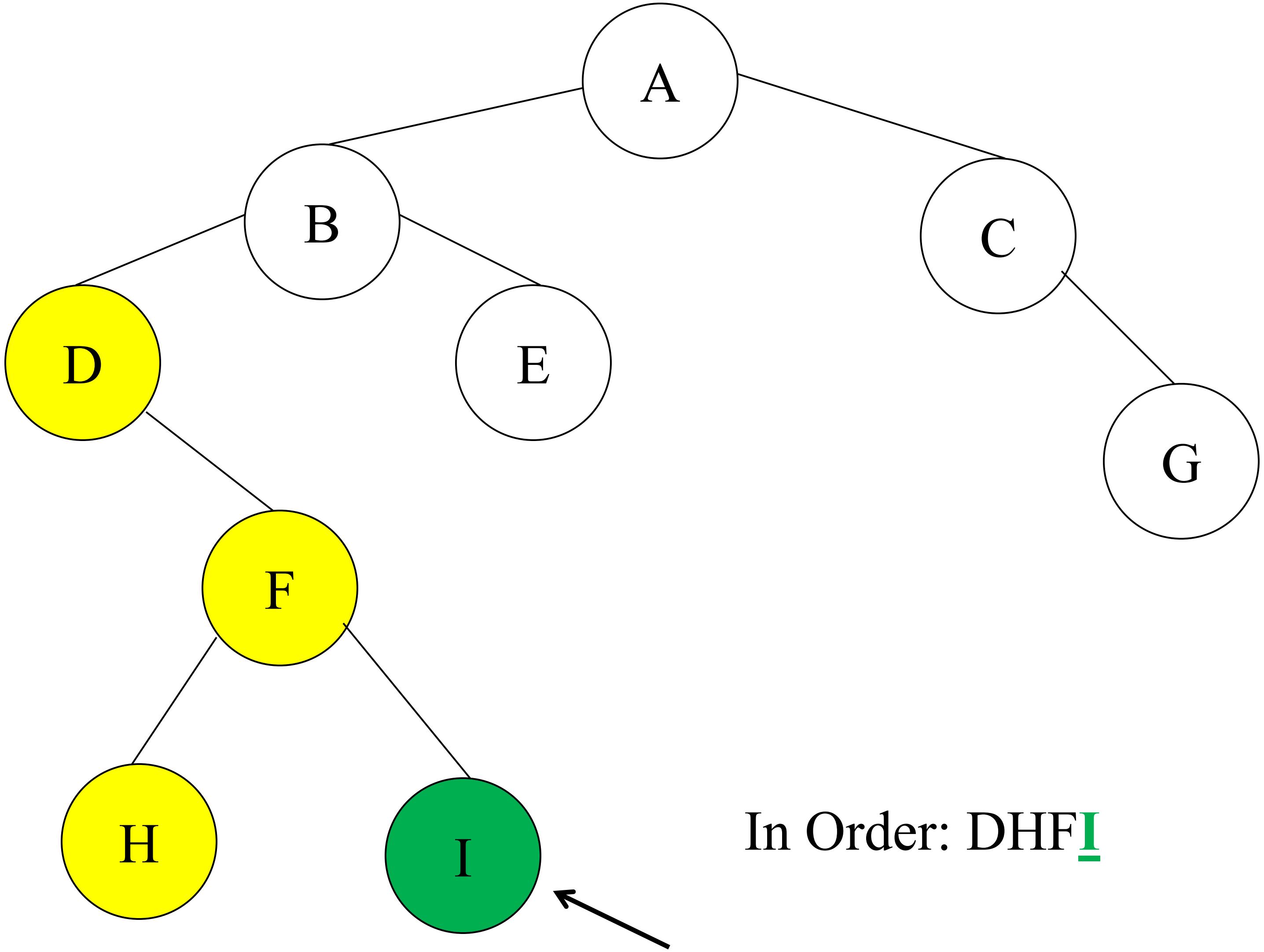

In Order: D**H**

# Tree Traversals: Another Example



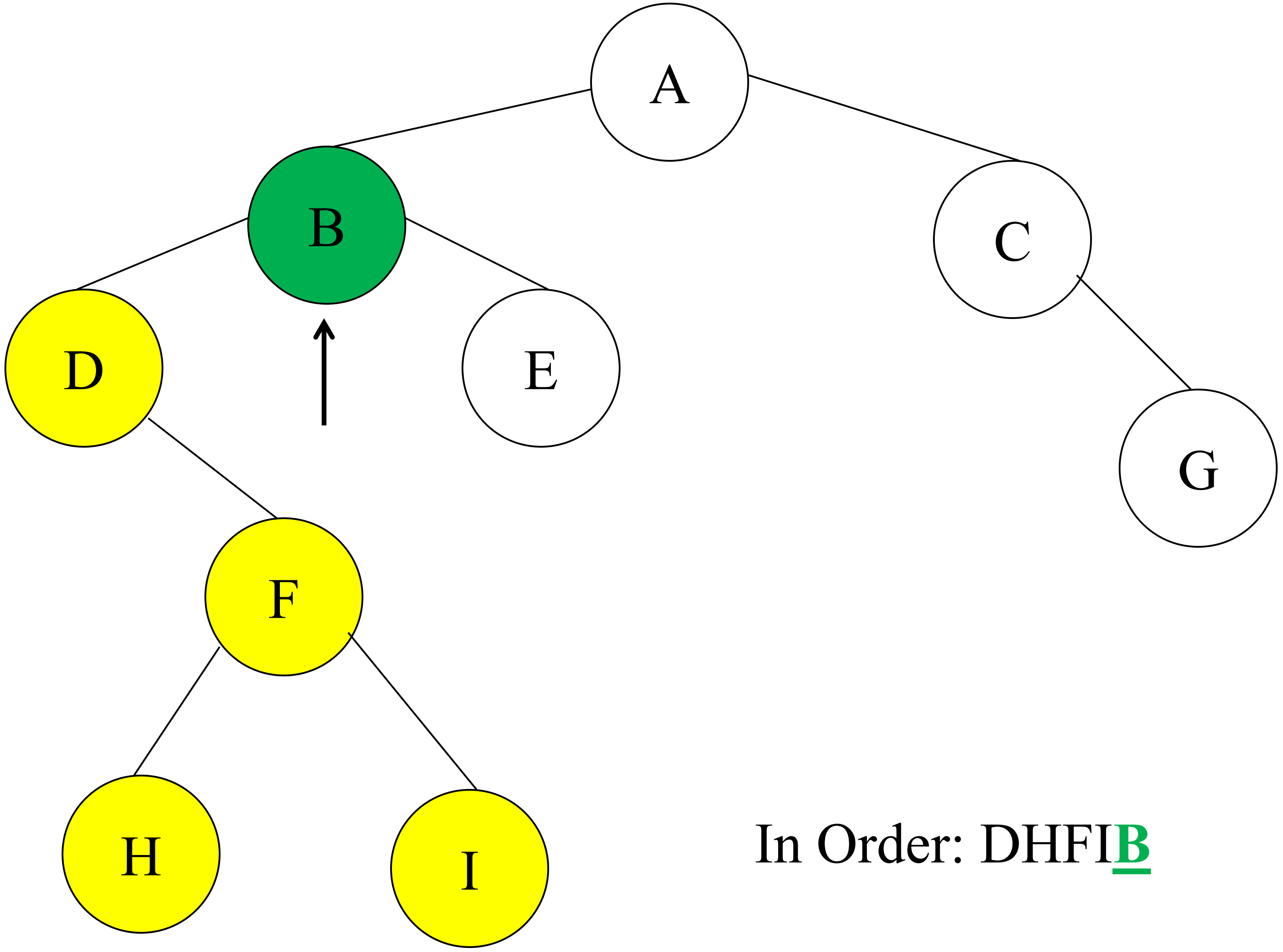

In Order: DH**F**

# Tree Traversals: Another Example



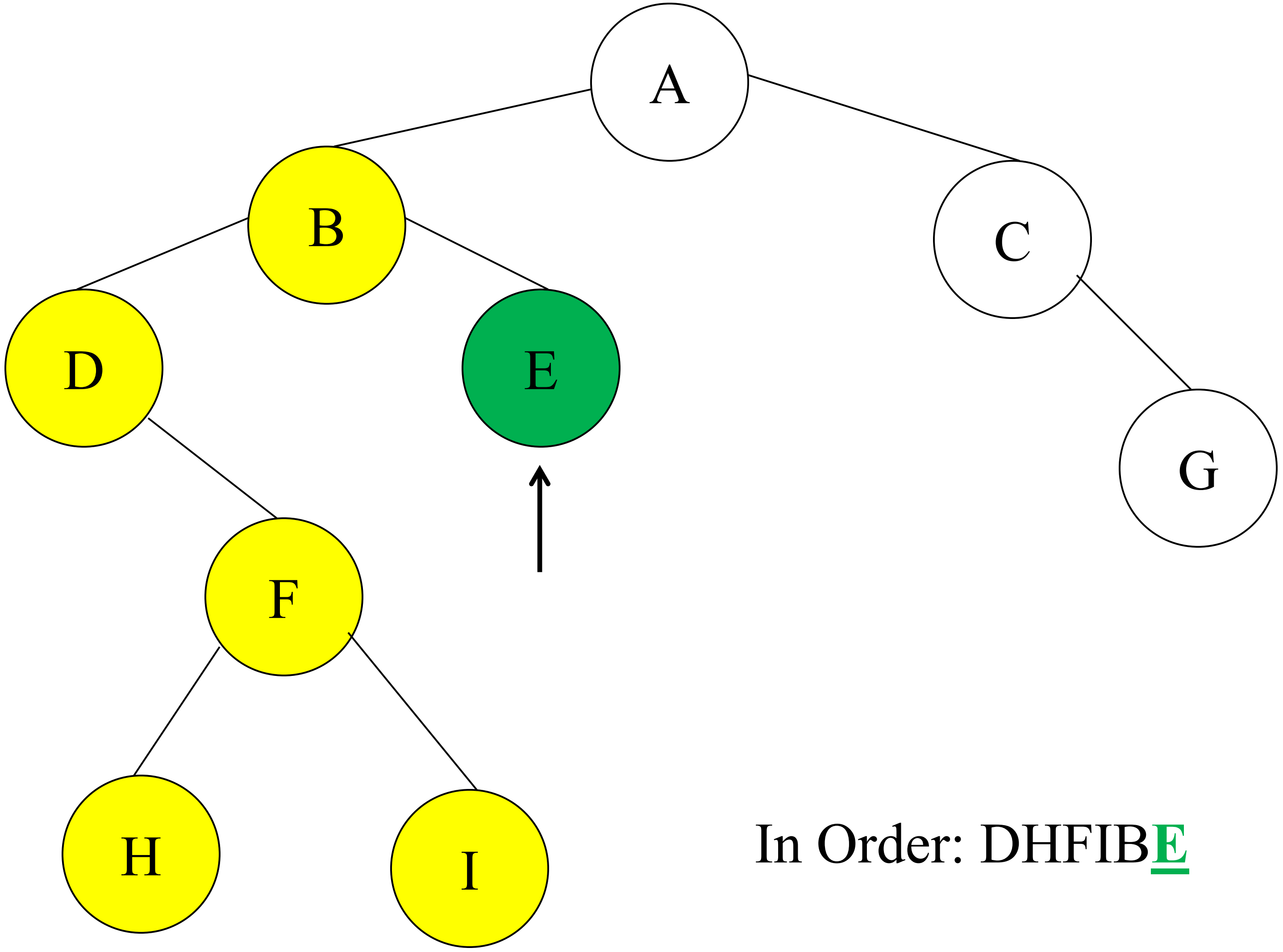

In Order: DHF**I**

# Tree Traversals: Another Example



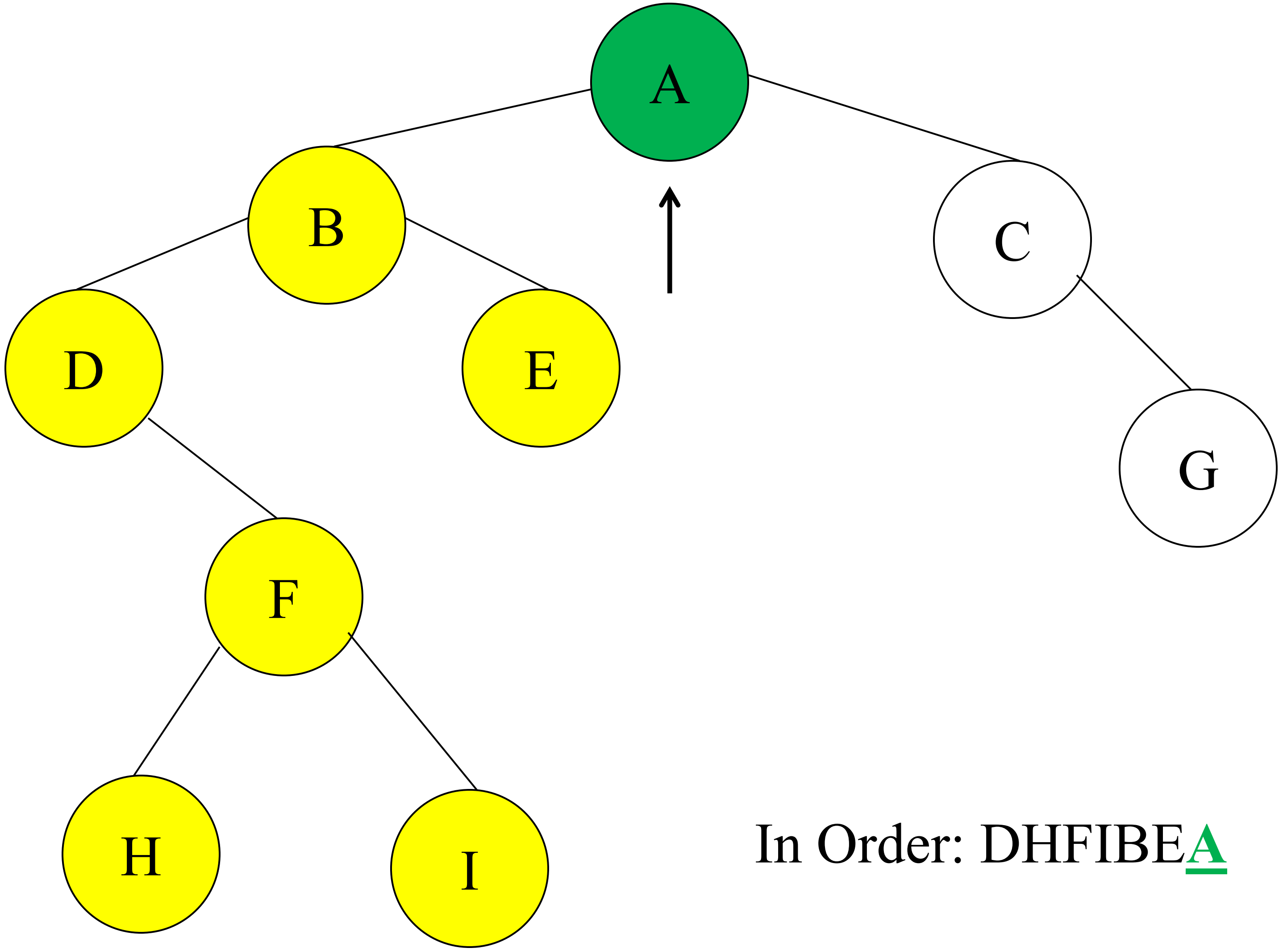

In Order: DHFI**B**

# Tree Traversals: Another Example



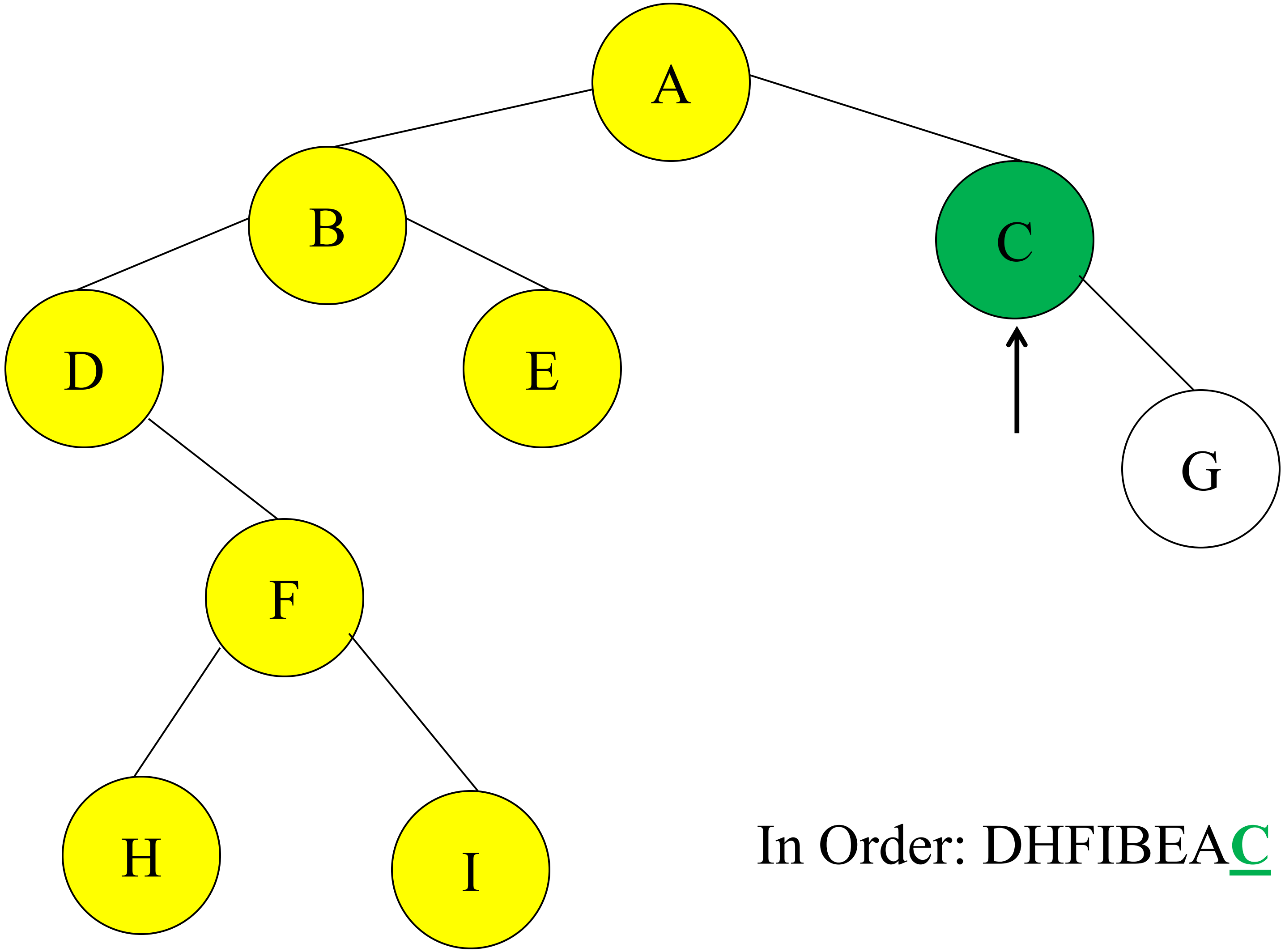

In Order: DHFIB**E**

# Tree Traversals: Another Example



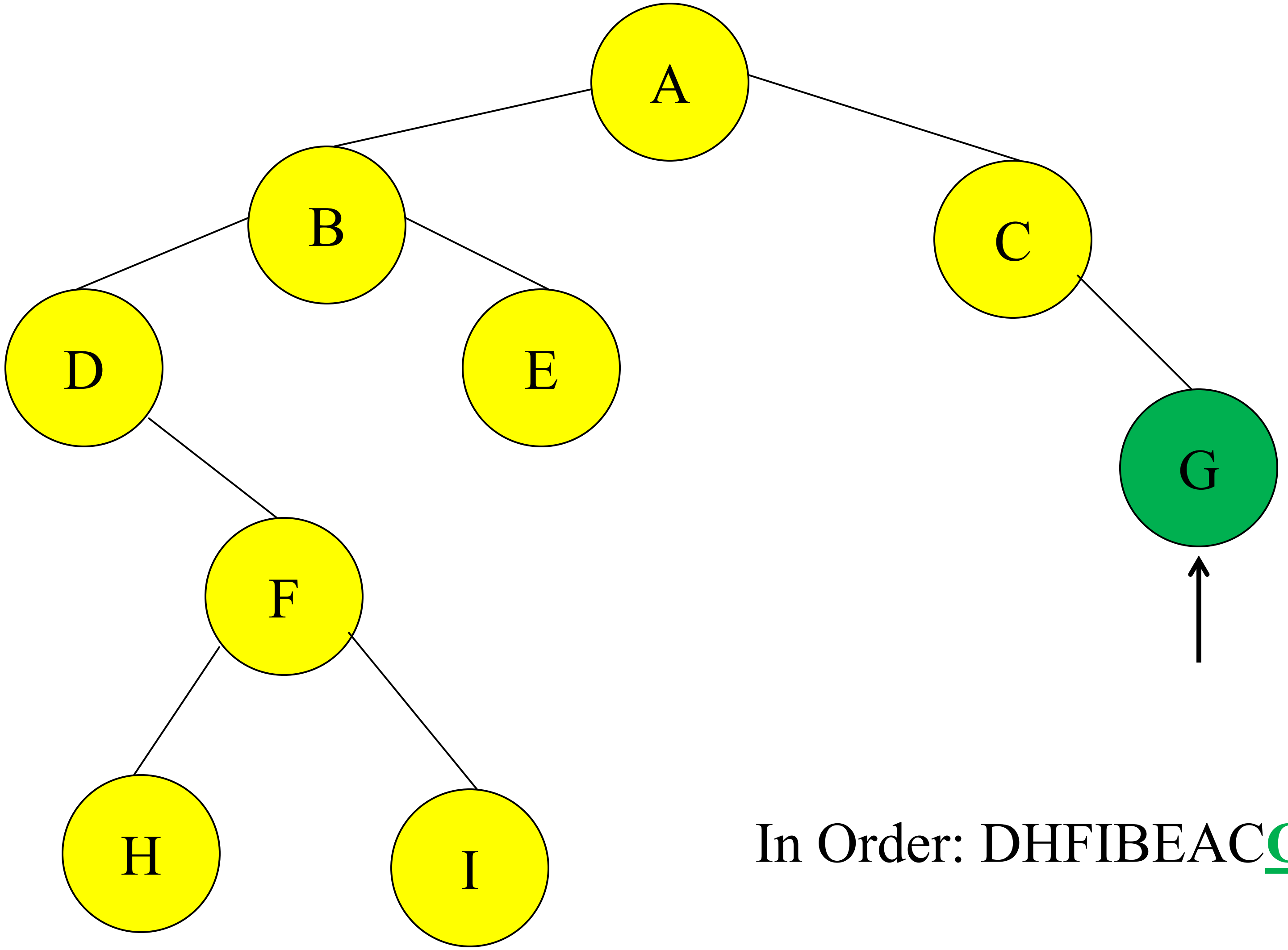

In Order: DHFIBEA

# Tree Traversals: Another Example



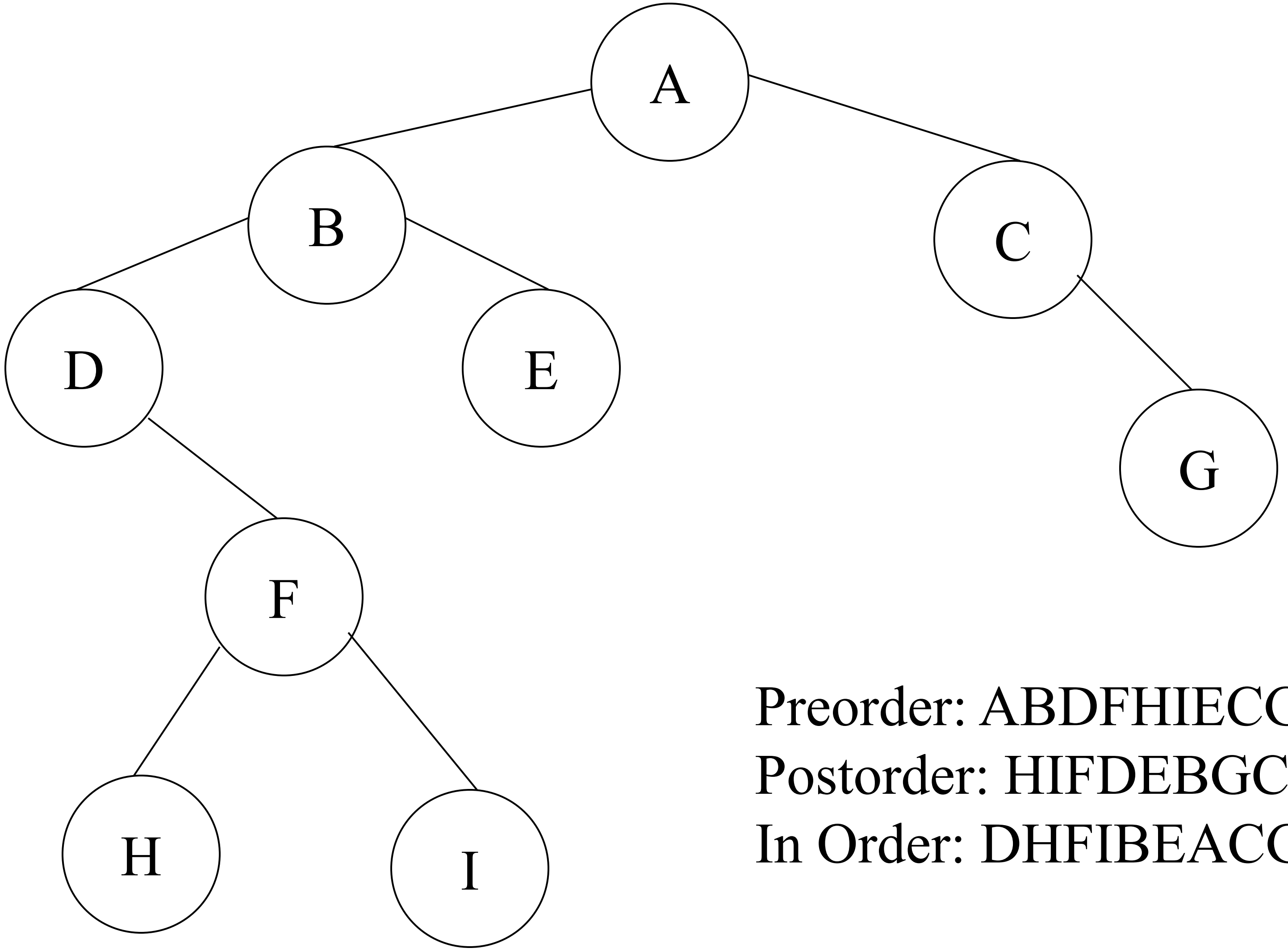

In Order: DHFIBEA**C**

# Tree Traversals: Another Example




In Order: DHFIBEAC**G**

# Tree Traversals: Another Example




Preorder: ABDFHIECG
Postorder: HIFDEBGCA
In Order: DHFIBEACG

# Binary Tree Implementation

# Implementation of a Binary Tree

- Implement a binary tree in Java

```
public class Node {
    private int data;
    private Node left;
    private Node right;

  public void setLeft(Node p)  {
    left = p;
  }

  public void setRight(Node p)  {
    right = p;
  }

  public int getData()  {
    return data;
  }

  public Node getLeft()  {
    return left;
  }

    public Node getRight()  {
    return right;
  }

  public void setData(int x)  {
    data = x;
  }
}
```

```java
public class Tree {
    private Node root;

    // tree() - The default constructor - Starts
    //          the tree as empty

    public Tree() {
        root = null;
    }

    // Tree() - An initializing constructor that
    //          creates a node and places in it
    //          the initial value

    public Tree(int x) {
        root = new Node();
        root.setData(x);
        root.setLeft(null);
        root.setRight(null);
    }

    public Node getRoot() {
        return root;
    }

    // newNode() -  Creates a new node with a
    //              zero as data by default
    public Node newNode() {
        Node p = new Node();
        p.setData(0);
        p.setLeft(null);
        p.setRight(null);
        return(p);
    }
```

```java
// newNode() -  Creates a new node with the
//              parameter x as its value
public Node newNode(int x)  {
    Node p = new Node();
    p.setData(x);
    p.setLeft(null);
    p.setRight(null);
    return(p);
}

//travTree() – initializes recursive
//   traversal of tree
 public void  travTree() {
    if (root != null)
      travSubtree(root);
    System.out.println();
 }
//travSubtree() – recursive method used to
 //   traverse a binary tree (inorder)
public void  travSubtree(Node p) {
    if (p != null)  {
        travSubtree(p.getLeft());
        System.out.print(p.getData() + "\t");
        travSubtree(p.getRight());
    }
}

// addLeft() -  Inserts a new node containing
//              x as the left child of p
public void  addLeft(Node p, int x) {
    Node   q = newNode(x);
    p.setLeft(q);
}

// addRight() -  Inserts a new node containing
//               x as the right child of p
public void  addRight(Node p, int x) {
    Node   q = newNode(x);
    p.setRight(q);
}
}
```

# Binary Tree Traversal: Recursion

- **preorder(n)**:
  - Visit the root,
  - traverse the left subtree (preorder(n)) and
  - then traverse the right subtree (preorder(n))
- **postorder(n)**:
  - Traverse the left subtree (postorder(n)),
  - traverse the right subtree (postorder(n)) and
  - then visit the root.
- **inorder**:
  - Traverse the left subtree (inorder(n)),
  - visit the root and
  - the traverse the right subtree (inorder(n)).

# Preorder traversal

- In preorder, the root is visited *first*
- Here's a preorder traversal to print out all the elements in the binary tree:

```
public void preorderPrint(BinaryTree bt) {
    if (bt == null) return;
    System.out.println(bt.value);
    preorderPrint(bt.leftChild);
    preorderPrint(bt.rightChild);
}
```

# Inorder traversal

- In inorder, the root is visited *in the middle*
- Here's an inorder traversal to print out all the elements in the binary tree:

```
public void inorderPrint(BinaryTree bt) {
    if (bt == null) return;
    inorderPrint(bt.leftChild);
    System.out.println(bt.value);
    inorderPrint(bt.rightChild);
}
```

# Postorder traversal

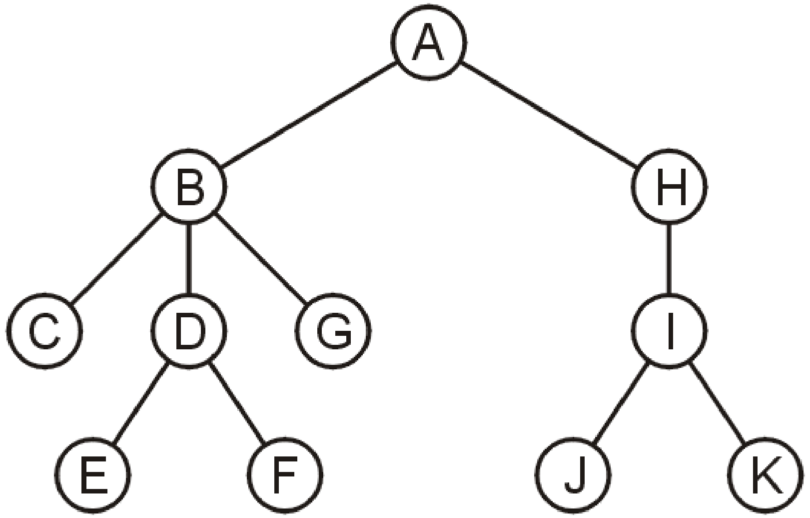
- In postorder, the root is visited *last*
- Here's a postorder traversal to print out all the elements in the binary tree:

```
public void postorderPrint(BinaryTree bt) {
    if (bt == null) return;
    postorderPrint(bt.leftChild);
    postorderPrint(bt.rightChild);
    System.out.println(bt.value);
}
```

# Breadth First Search using Queue

# Application

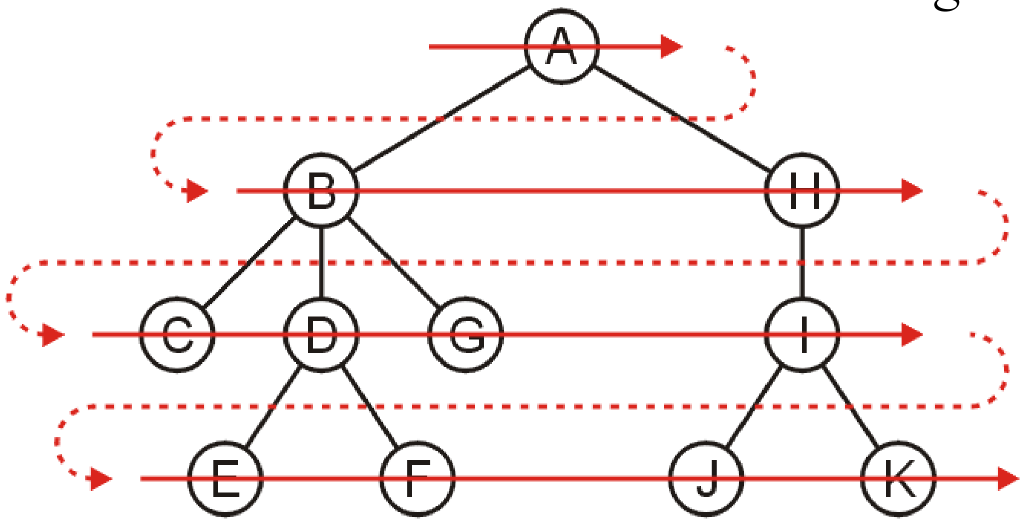
- Another application is performing a breadth-first traversal of a directory tree
  - Consider searching the directory structure



# Application

- We would rather search the more shallow directories first then plunge deep into searching one sub-directory and all of its contents

- One such search is called a *breadth-first traversal*
  - Search all the directories at one level before descending a level



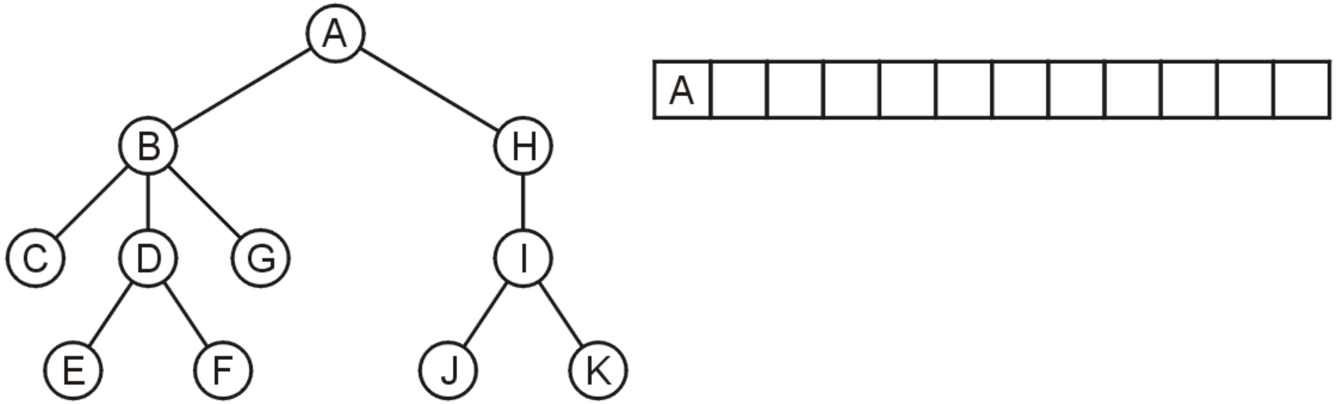
# Application

- The easiest implementation is:
  - Place the root directory into a queue
  - While the queue is not empty:
    - Pop the directory at the front of the queue
    - Push all of its sub-directories into the queue

- The order in which the directories come out of the queue will be in breadth-first order

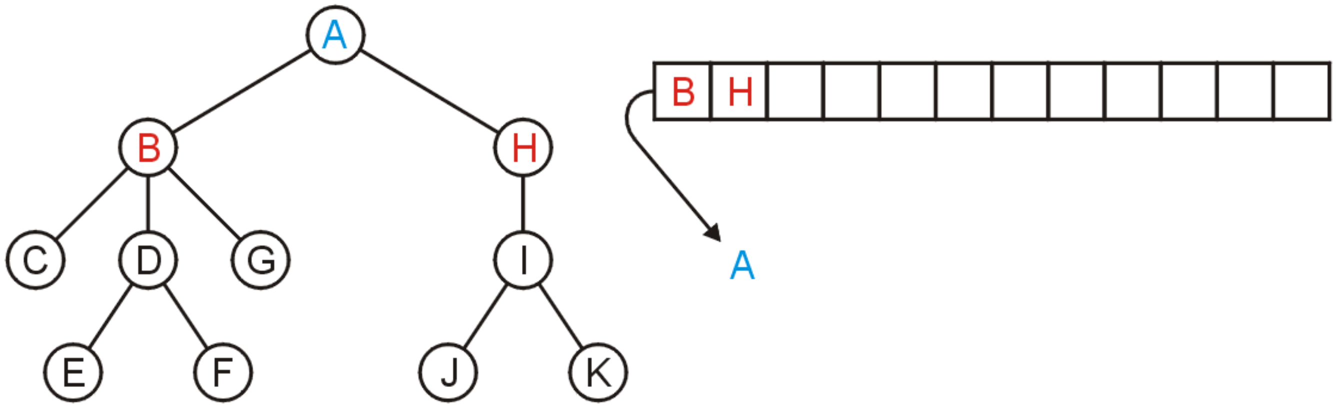
# Application

- Push the root directory A



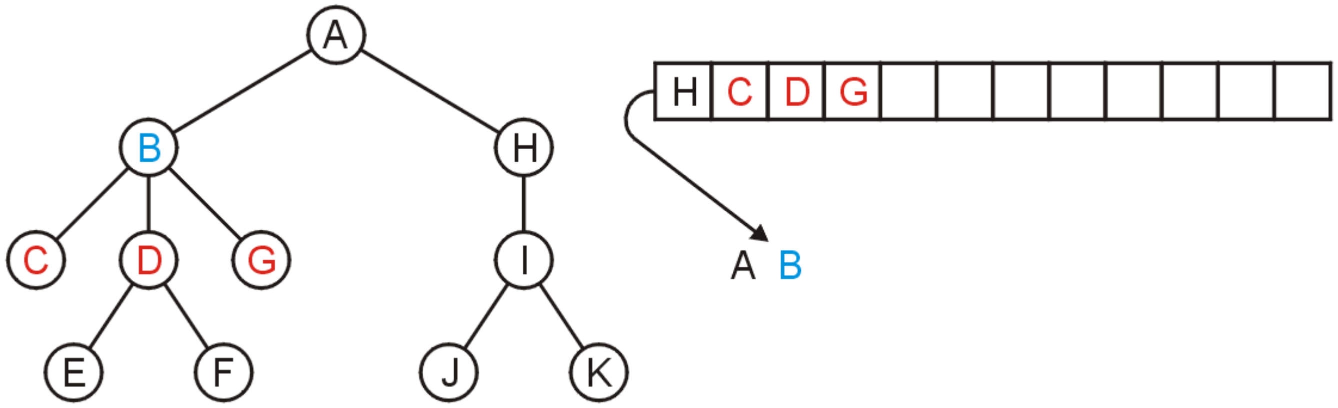
# Application

- Pop A and push its two sub-directories:  B and H



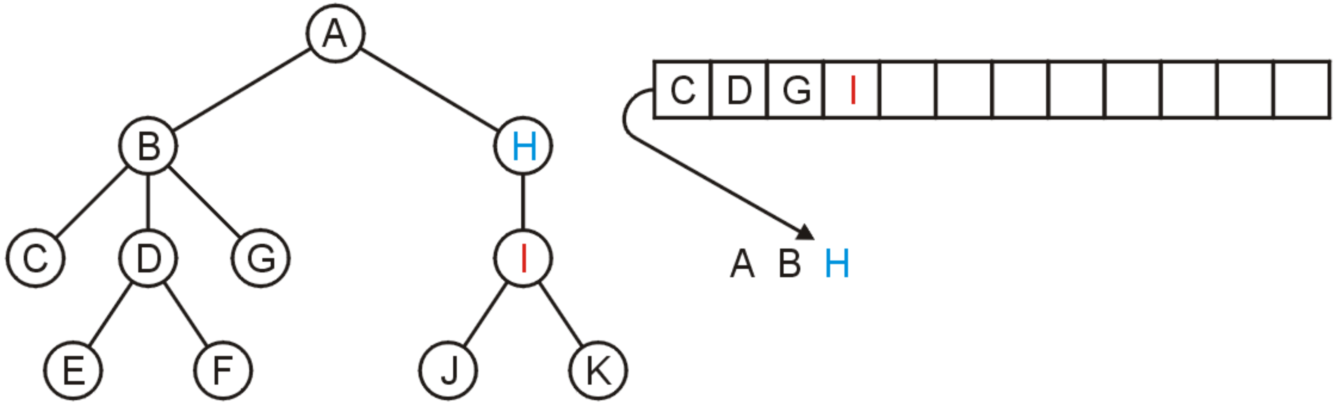
# Application

- Pop B and push C, D, and G



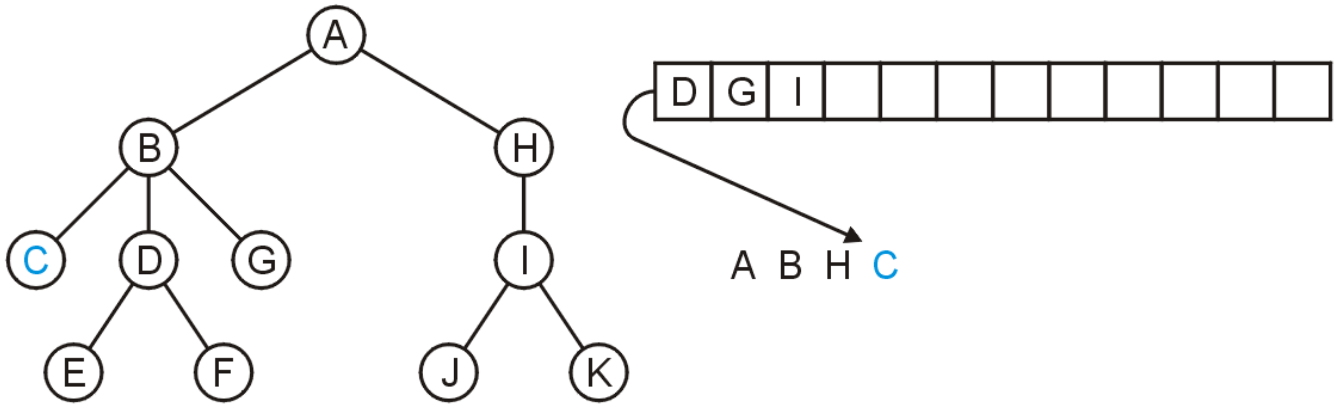
# Application

- Pop H and push its one sub-directory I



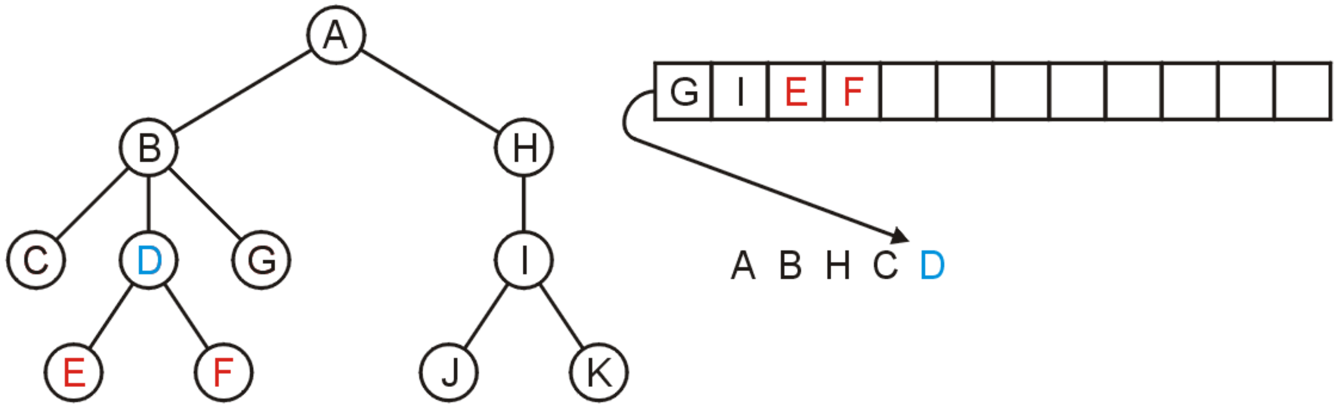
# Application

- Pop C:  no sub-directories



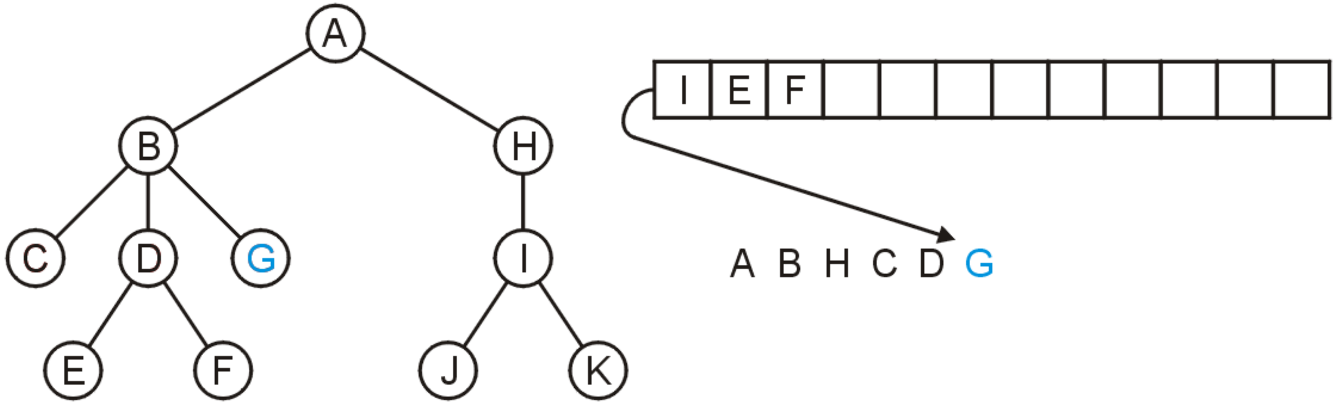
# Application

- Pop D and push E and F



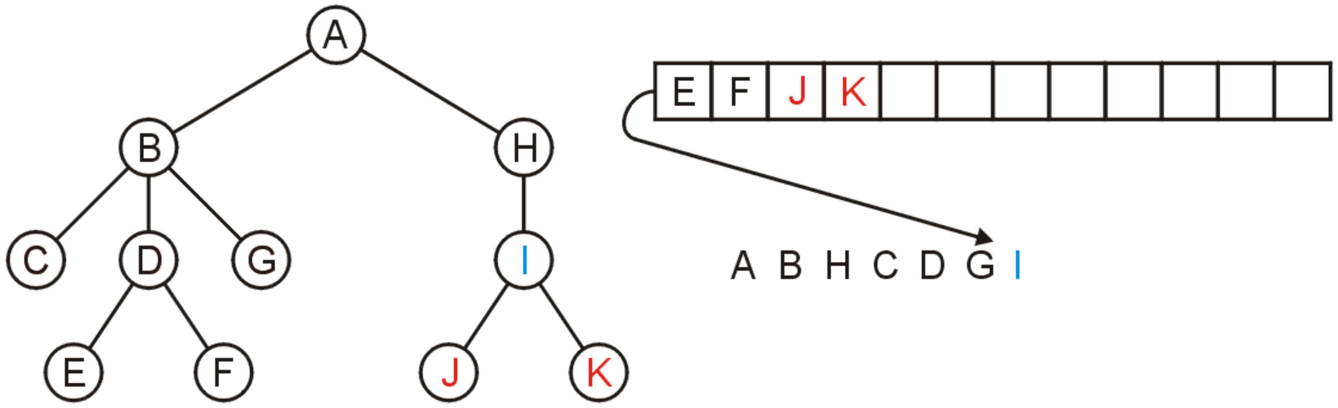
# Application

- Pop G



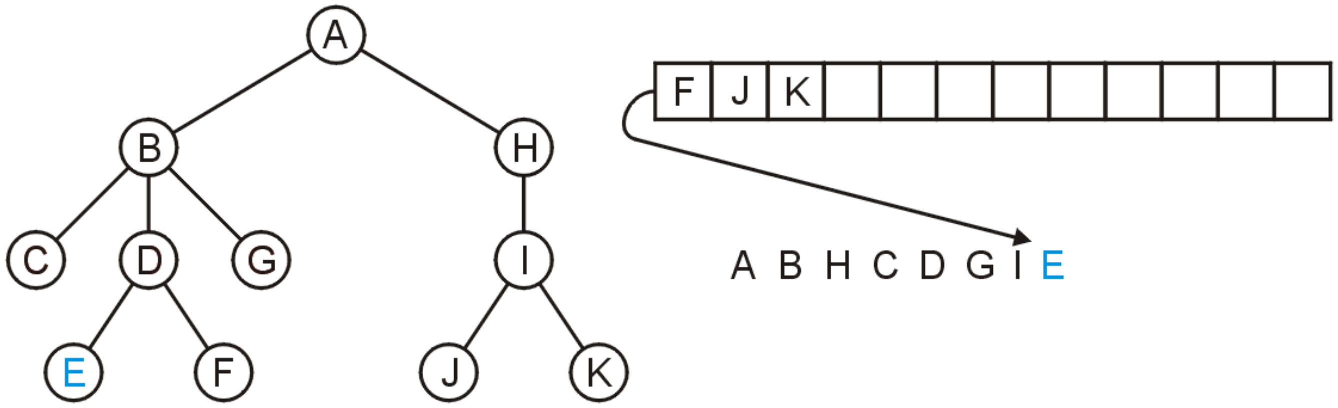
# Application

- Pop I and push J and K



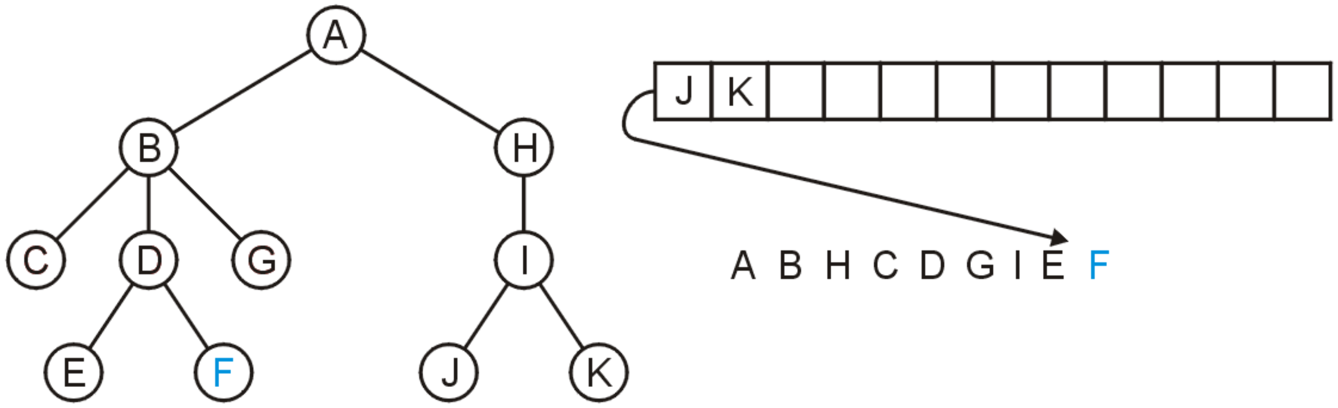
# Application

- Pop E



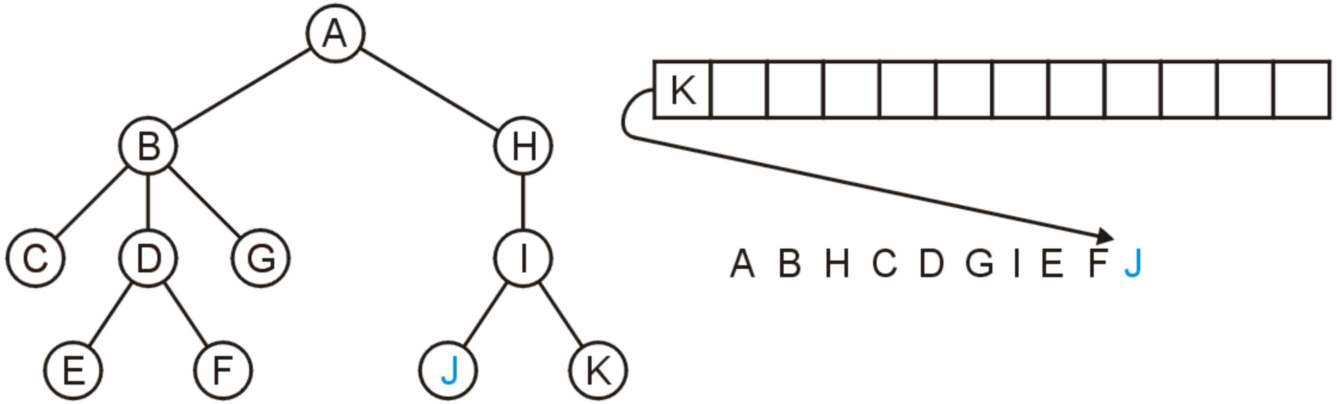
# Application

- Pop F



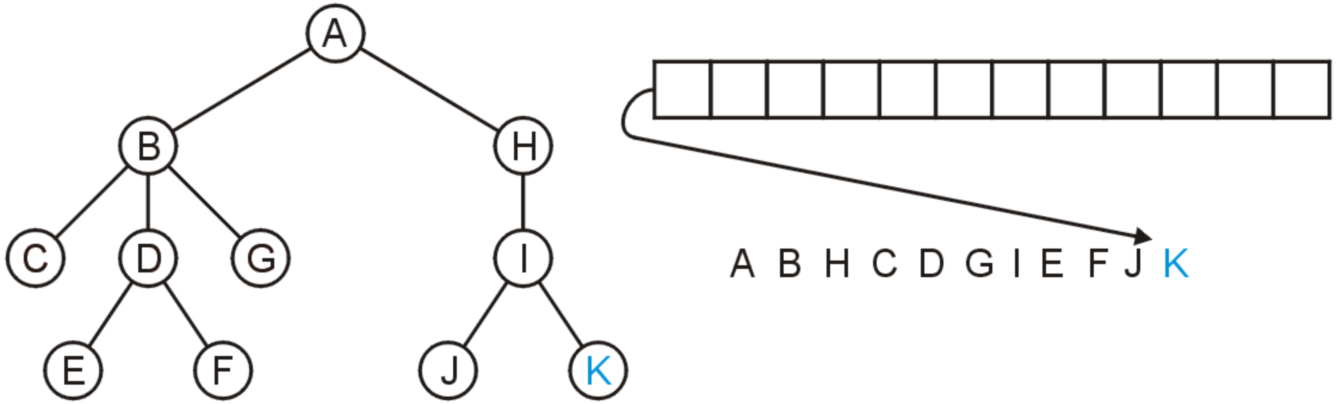
# Application

- Pop J



# Application

- Pop K and the queue is empty

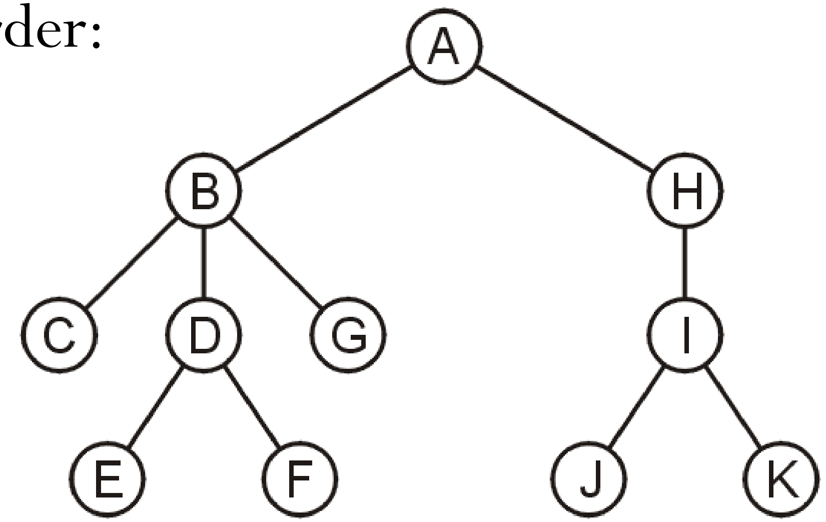


# Application

- The resulting order

  A B H C D G I E F J K

- is in breadth-first order:



# Summary

- Terminology used for the tree data structure:
  - root node, leaf node
  - parent node, children, and siblings
  - ordered trees
  - paths, depth, and height
  - ancestors, descendants, and subtrees
- XHTML and CSS Tree
- Binary Tree
- Traversal
  - BFS

# References


- Douglas Wilhelm Harder, Algorithms and Data Structures, Department of Electrical and Computer Engineering, University of Waterloo https://ece.uwaterloo.ca/~dwharder/aads/
- Donald E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, 3rd Ed., Addison Wesley, 1997.
- CSE326: Data Structure, Department of Computer Science and Engineering, University of Washington https://courses.cs.washington.edu/courses/cse326
- Mike Scott, CS 307 Fundamentals of Computer Science, https://www.cs.utexas.edu/~scottm/cs307/
- Debasis Samanta, Computer Science & Engineering, Indian Institute of Technology Kharagpur, Spring-2017, Programming and Data Structures. https://cse.iitkgp.ac.in/~dsamanta/courses/pds/index.html
- Weiss, Data Structures and Algorithm Analysis in C++, 3rd Ed., Addison Wesley, §3.3.1, p.75.
- Cormen, Leiserson, and Rivest, Introduction to Algorithms, McGraw Hill, 1990, §11.1, p.200.
- Weiss, Data Structures and Algorithm Analysis in C++, 3rd Ed., Addison Wesley, §3.3.1, p.75.
- Wikipedia, http://en.wikipedia.org/wiki/Double-ended_queue

תודה רבה
Hebrew

Ευχαριστώ
Greek

Спасибо
Russian

Danke
German

धन्यवादः
Sanskrit

Merci
French

ধন্যবাদ
Bangla

நன்றி
Tamil

شكرا
Arabic

ಧನ್ಯವಾದಗಳು
Kannada

*Thank You*
English

നന്ദി
Malayalam

Grazie
Italian

多謝
Traditional Chinese

ధన్యవాదాలు
Telugu

આભાર
Gujarati

ਧੰਨਵਾਦ
Punjabi

धन्यवाद
Hindi & Marathi

Gracias
Spanish

多谢
Simplified Chinese

https://sites.google.com/site/animeshchaturvedi07

Obrigado
Portuguese

ありがとうございました
Japanese

ขอบคุณ
Thai

감사합니다
Korean