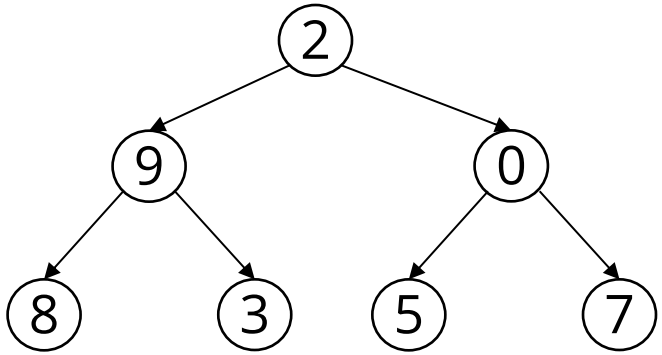
COSC 222 Data Structure

Trees

Binary Trees

- **Recursive definition** of a binary tree:
  it is
  - The empty tree
  - Or, a tree which has a root whose left and right subtrees are binary trees

Binary Tree



Left subtree,
which is also a binary tree

Right subtree
(also a binary tree)

Tree Traversals

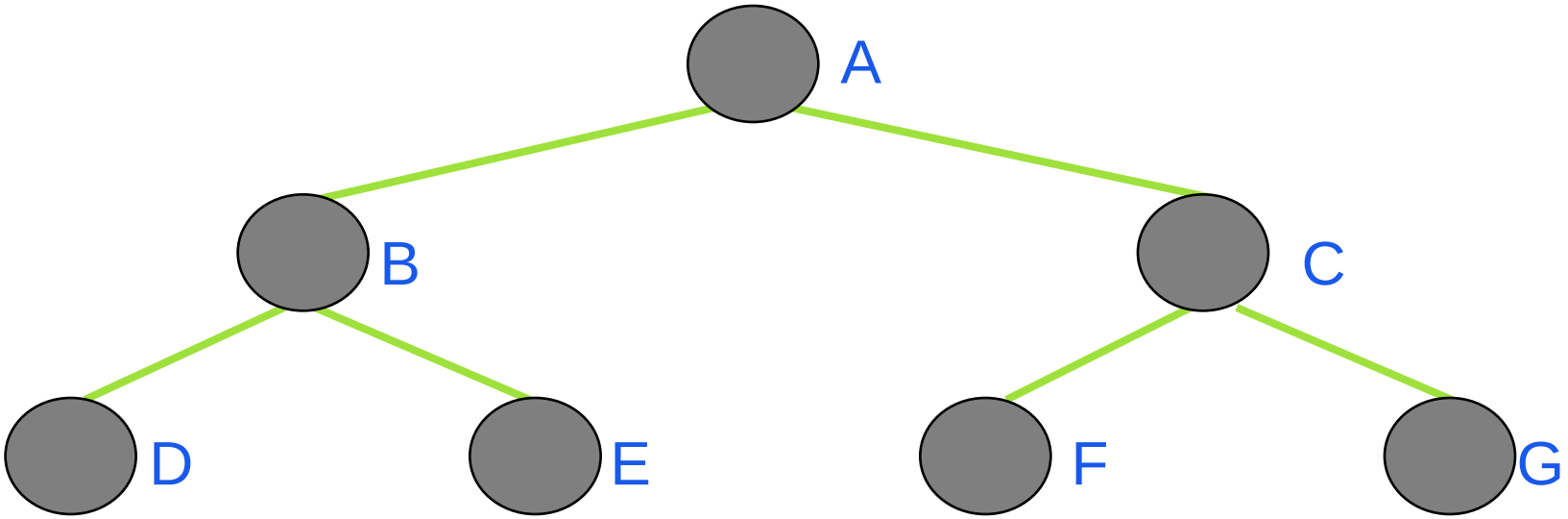
- A **traversal** of a tree requires that each node of the tree be visited once
  - Example: a typical reason to traverse a tree is to **display** the data stored at each node of the tree
- Standard traversal orderings:
  - Pre-order
  - In-order
  - Post-order
  - Level-order

Preorder Traversal

- Start at the root
- Visit each node, followed by its children; we will choose to visit left child before right

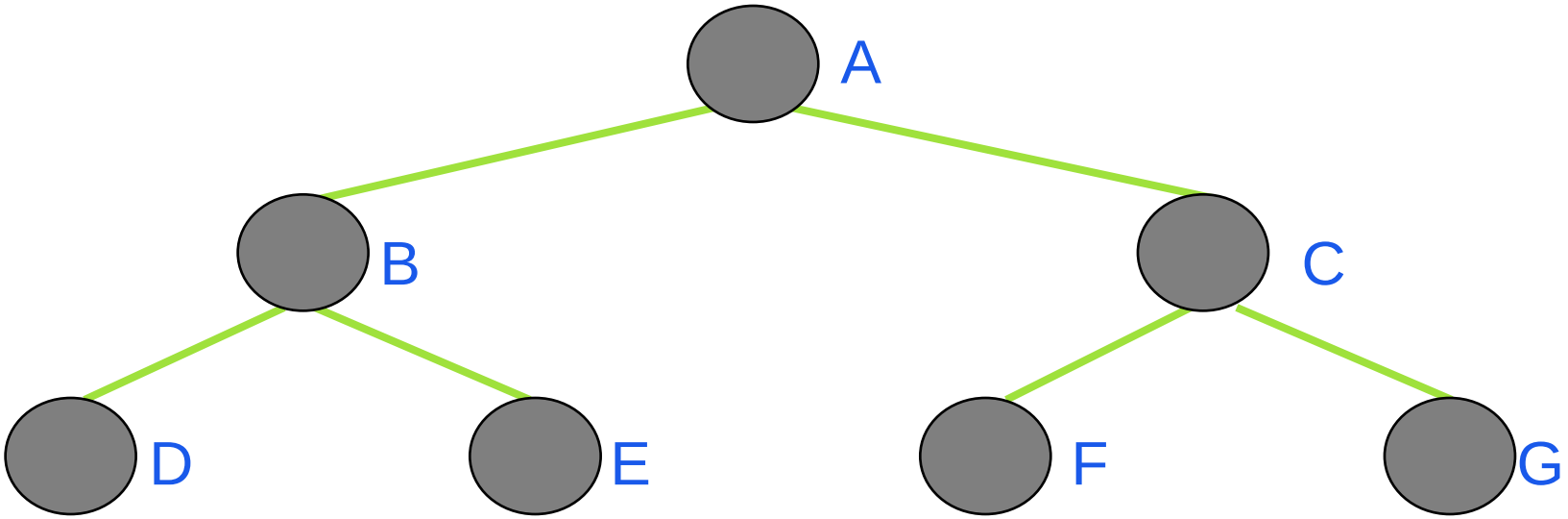
- Recursive algorithm for preorder traversal:
  - If tree is not empty,
    Visit root node of tree
    Recursively traverse left subtree
    Recursively traverse right subtree

Traversals



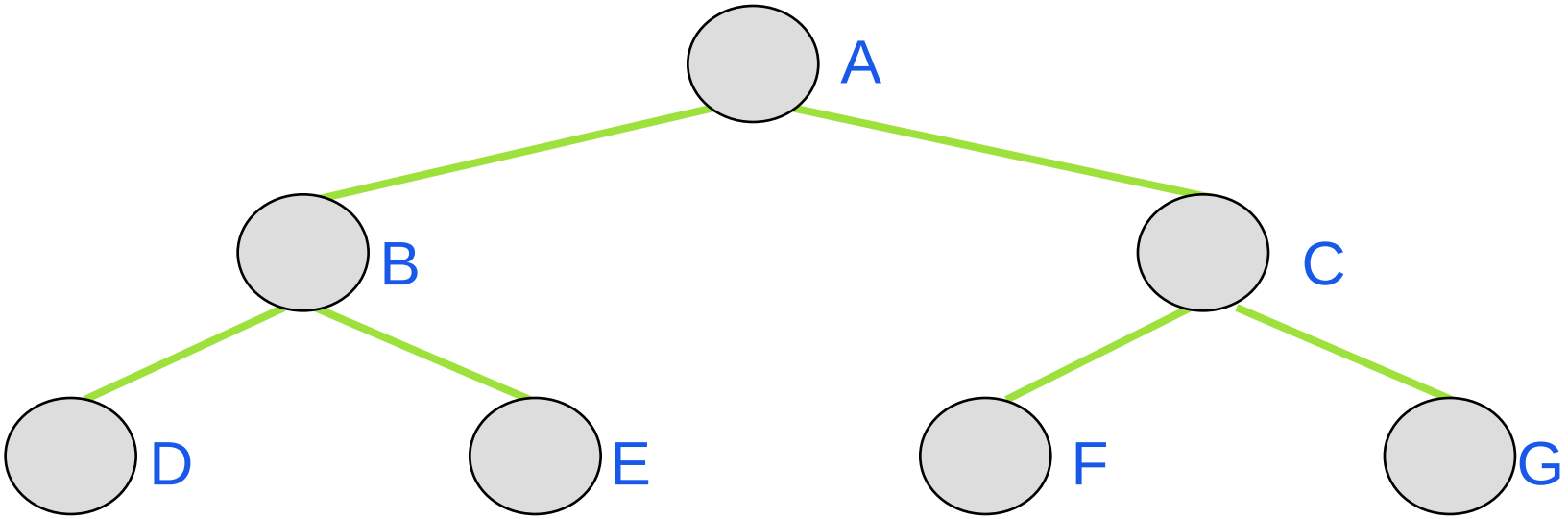
Output: ?

Preorder Traversal

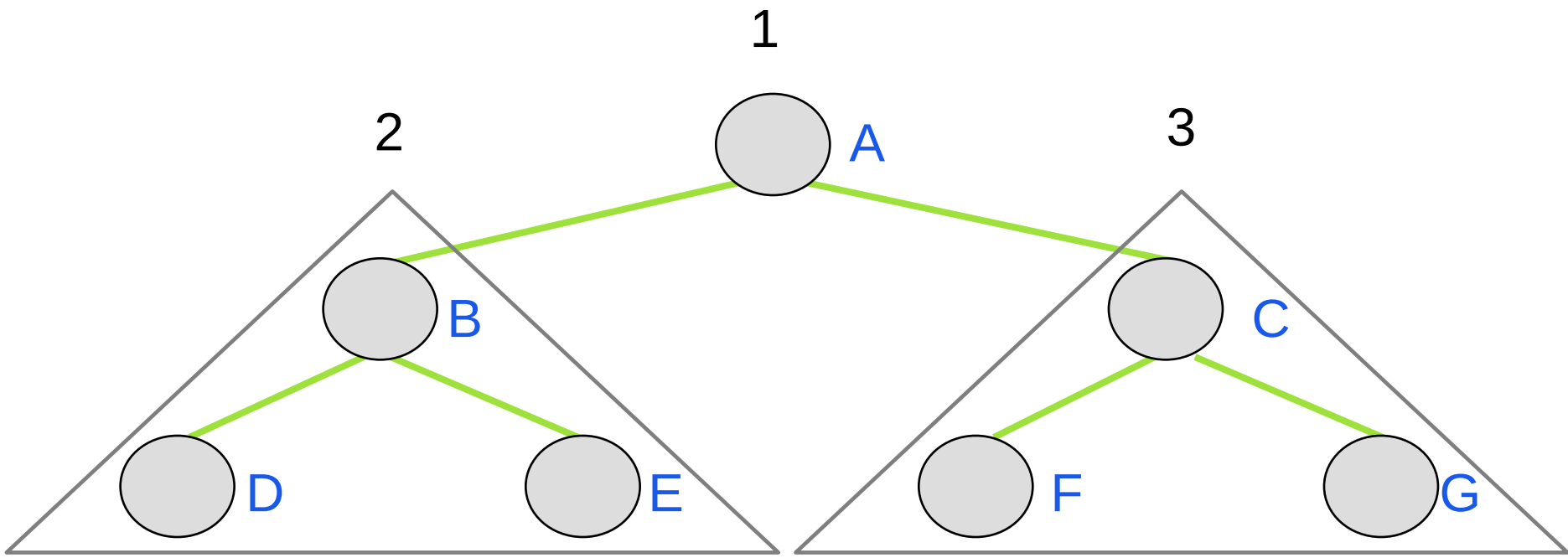


Output: A→B→D→E→C→F→G

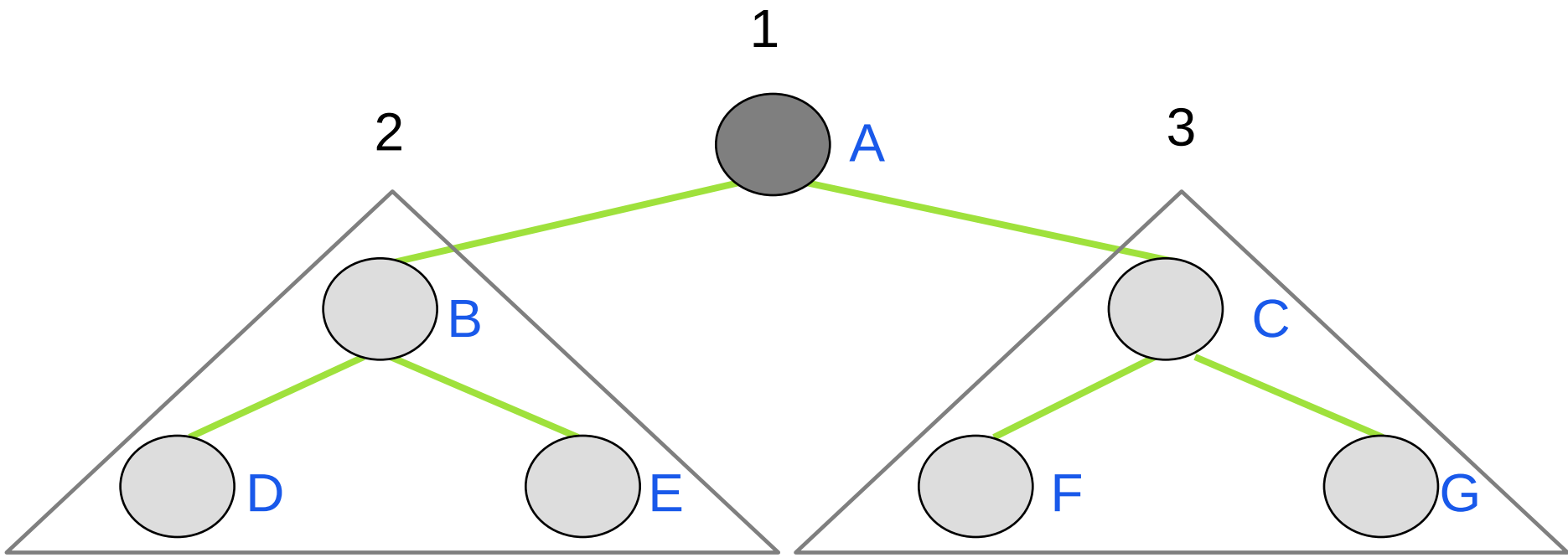
Preorder Traversal



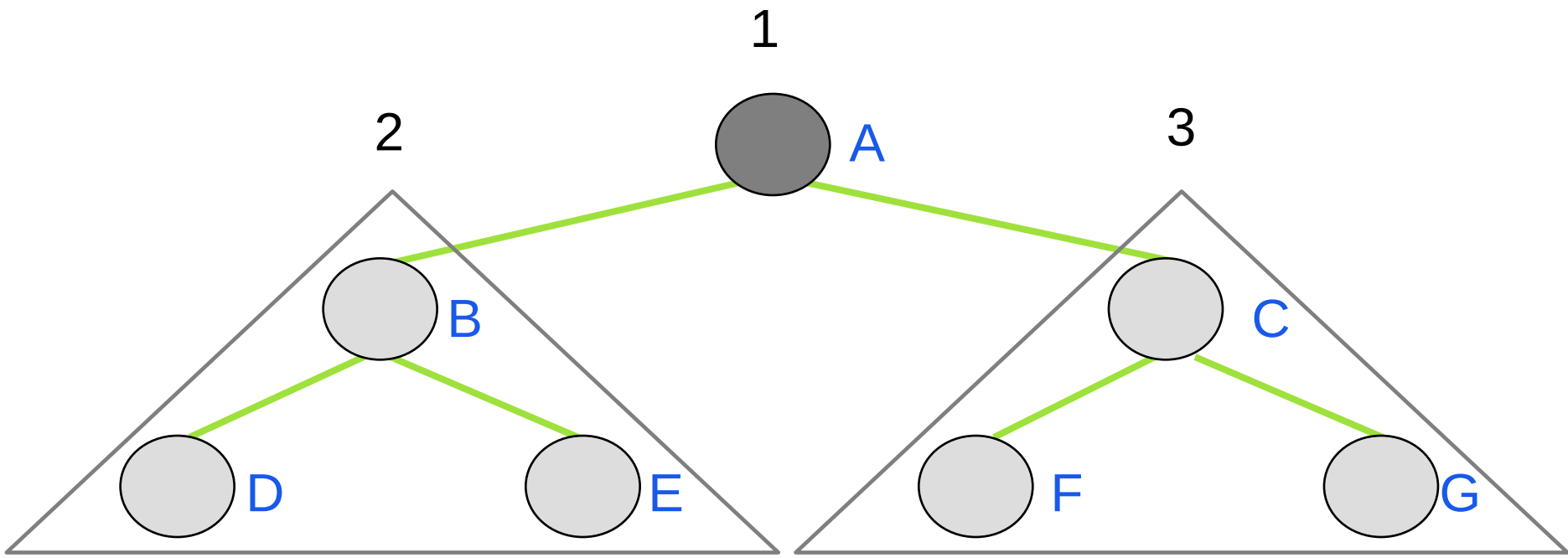
Preorder Traversal



Preorder Traversal

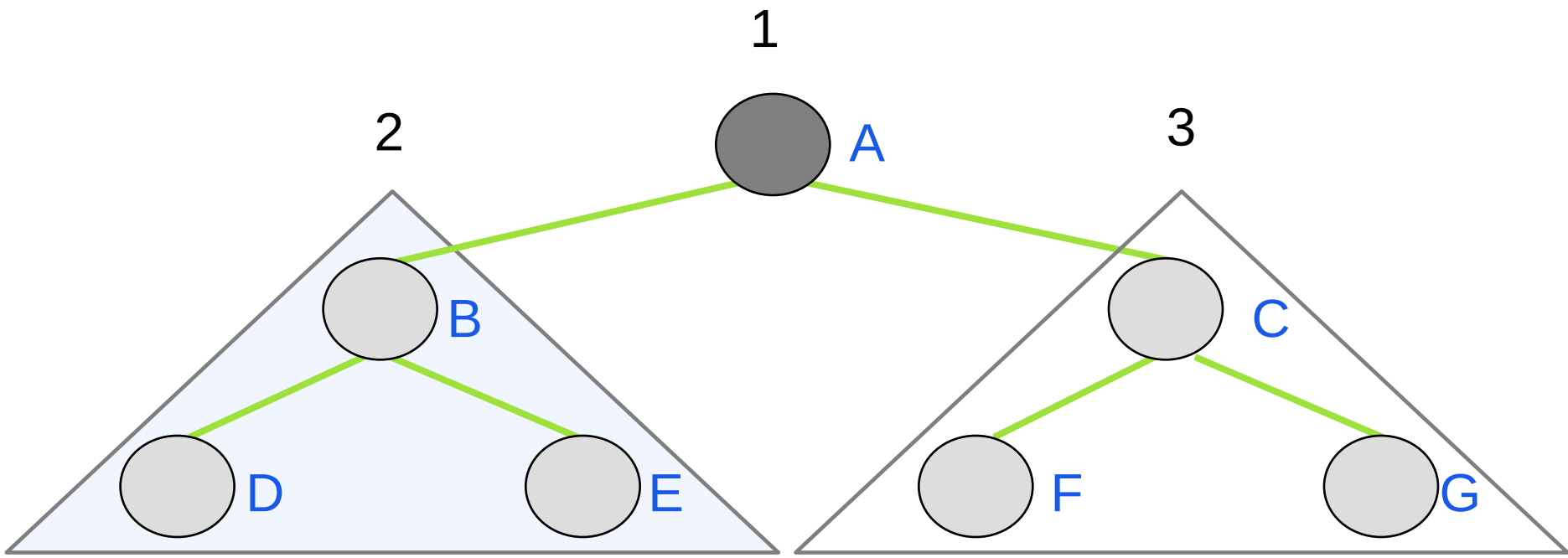


Preorder Traversal



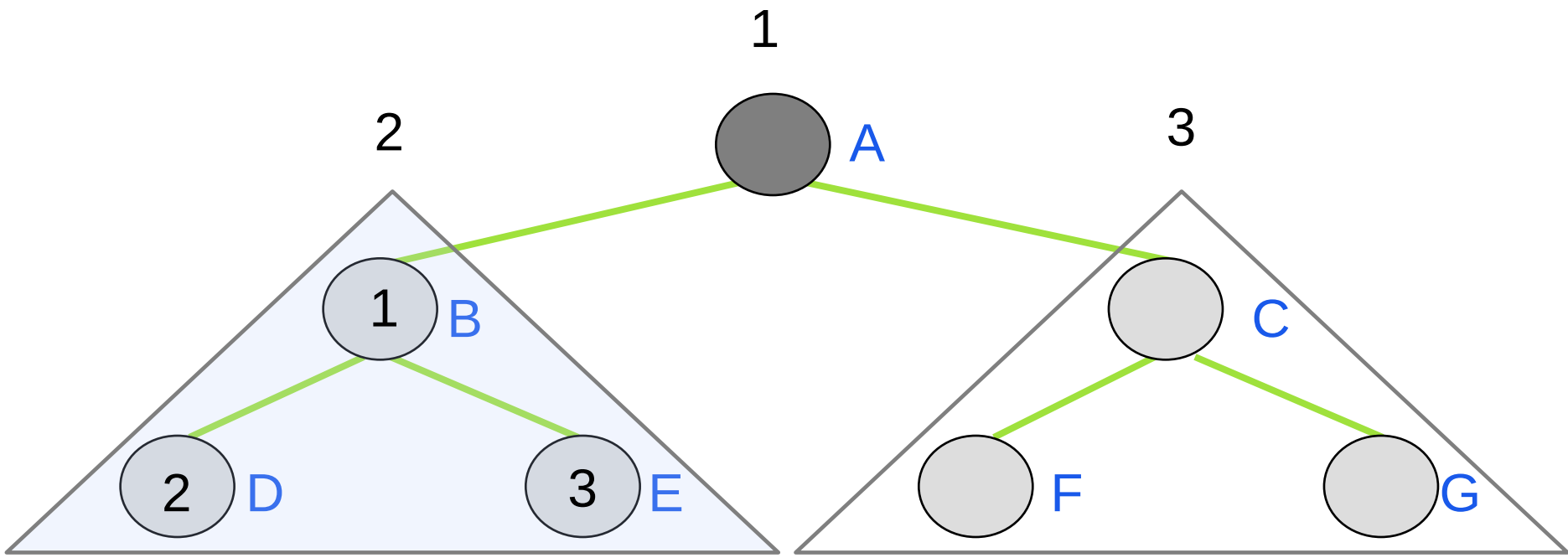
Output: A

Preorder Traversal



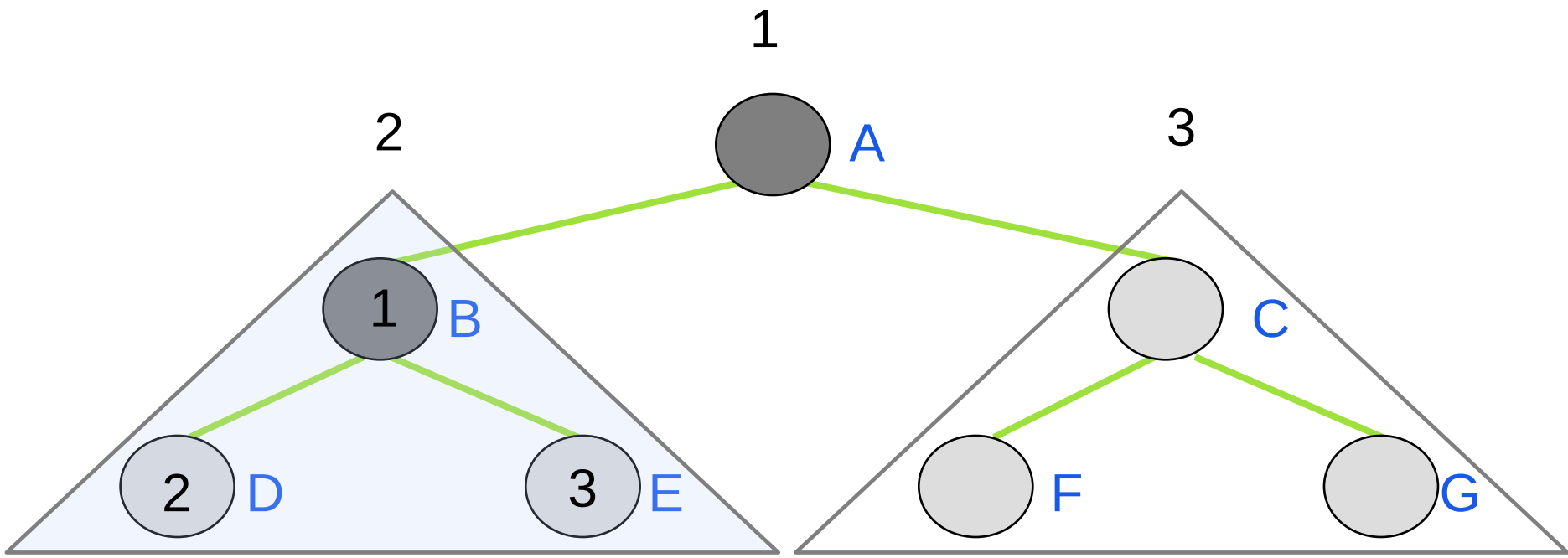
Output: A

Preorder Traversal



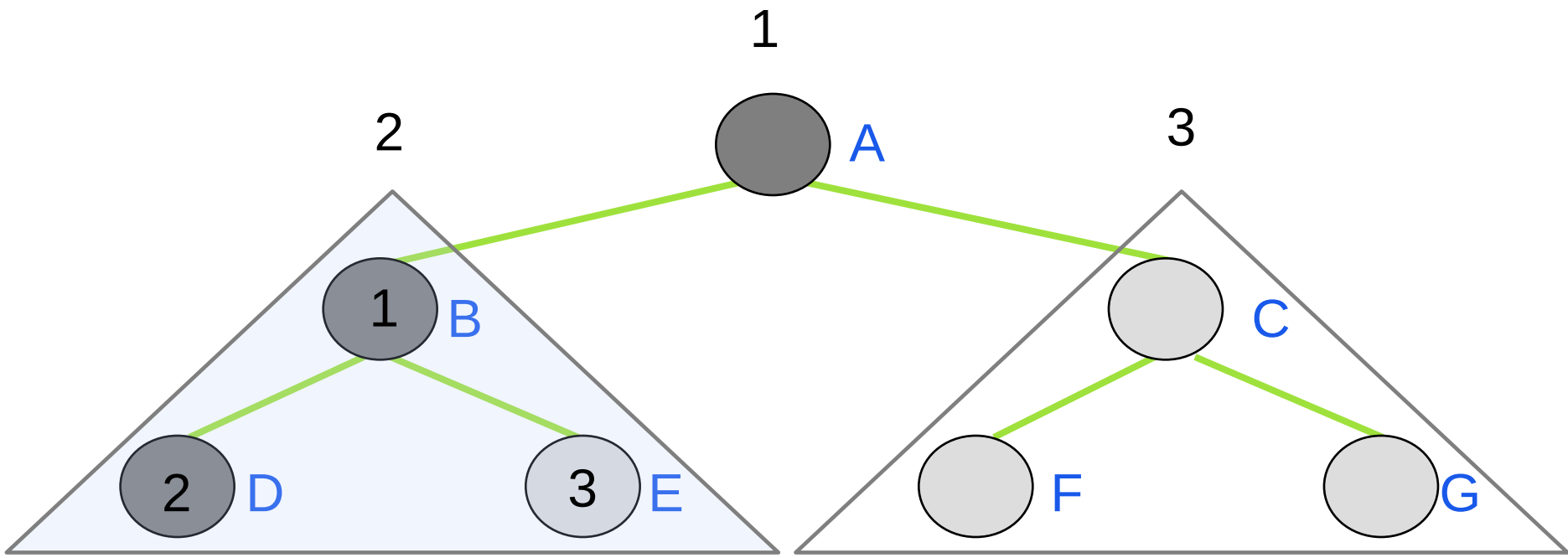
Output: A

Preorder Traversal



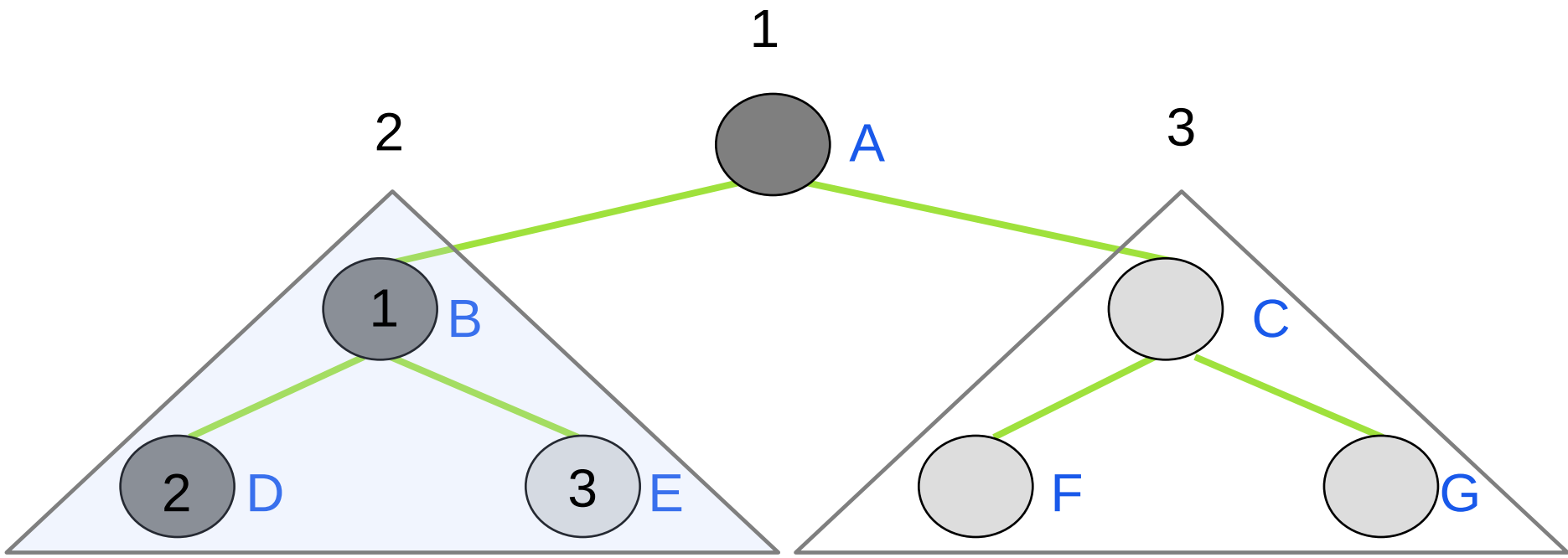
Output: A→B

Preorder Traversal



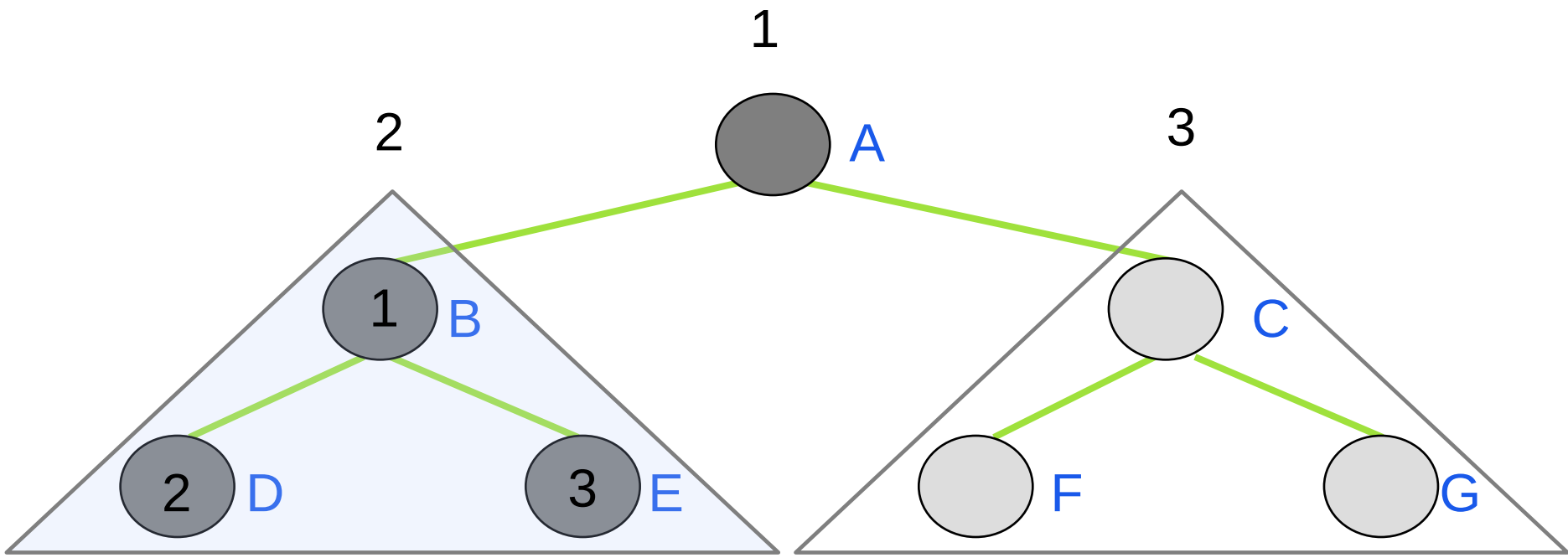
Output: A→B

Preorder Traversal



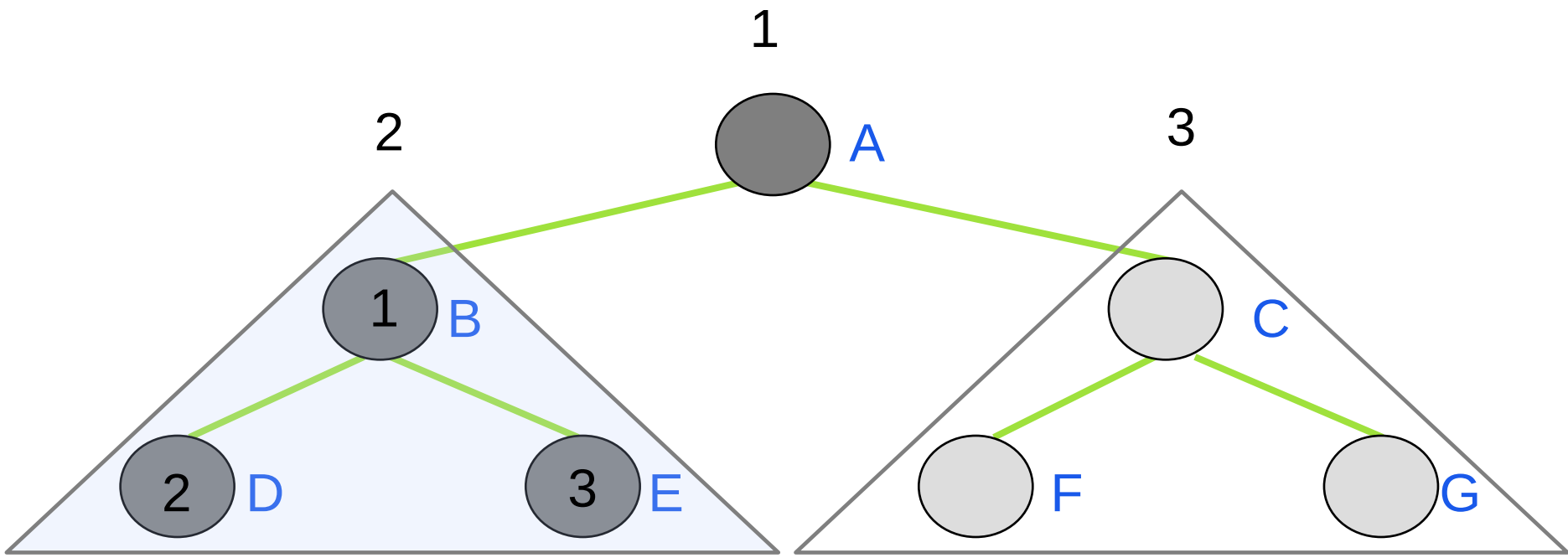
Output: A→B→D

Preorder Traversal



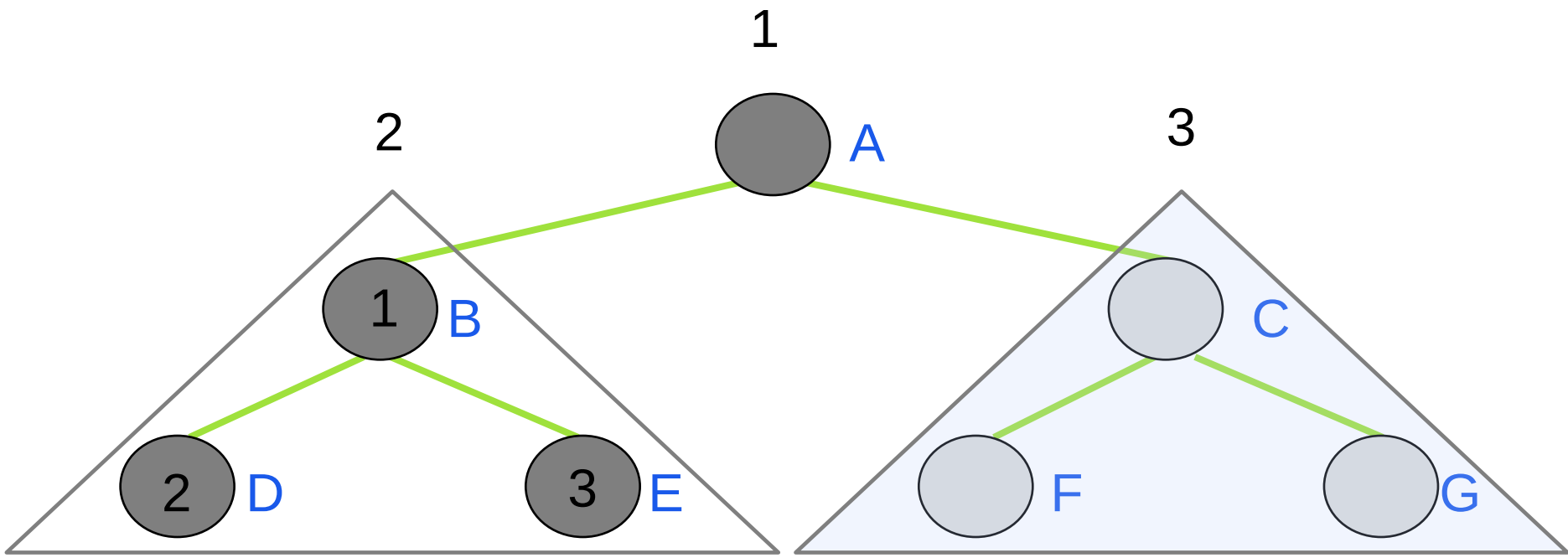
Output: A→B→D

Preorder Traversal



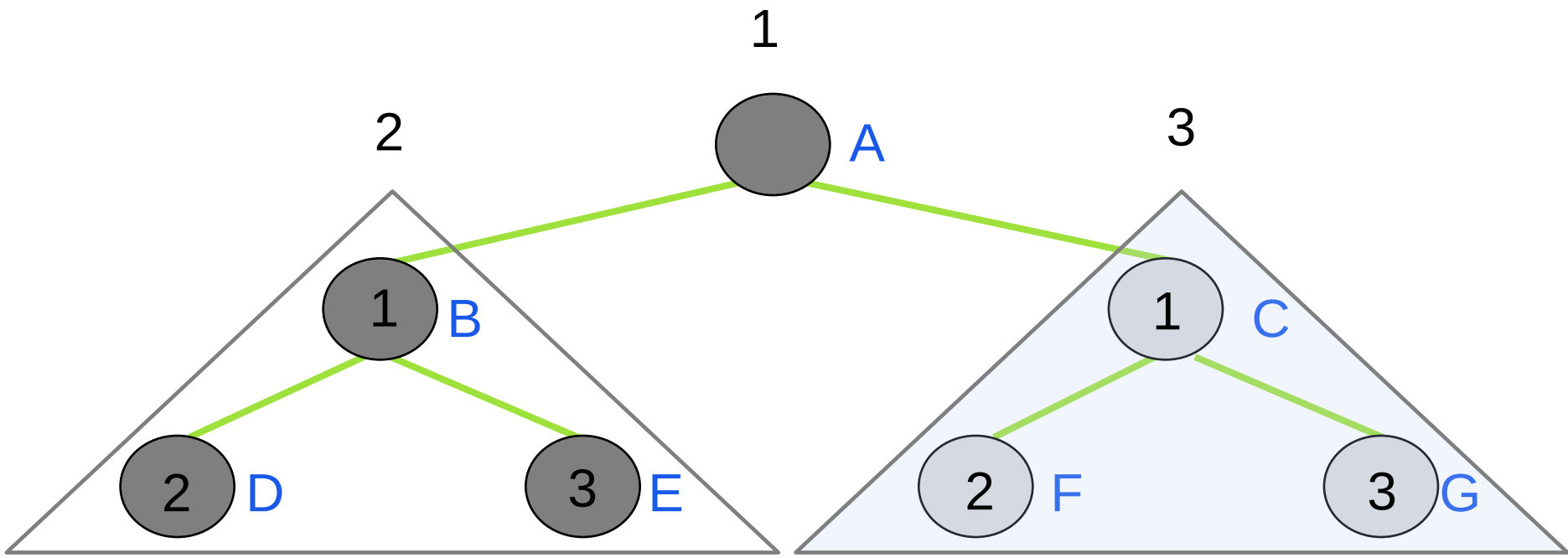
Output: A→B→D→E

Preorder Traversal



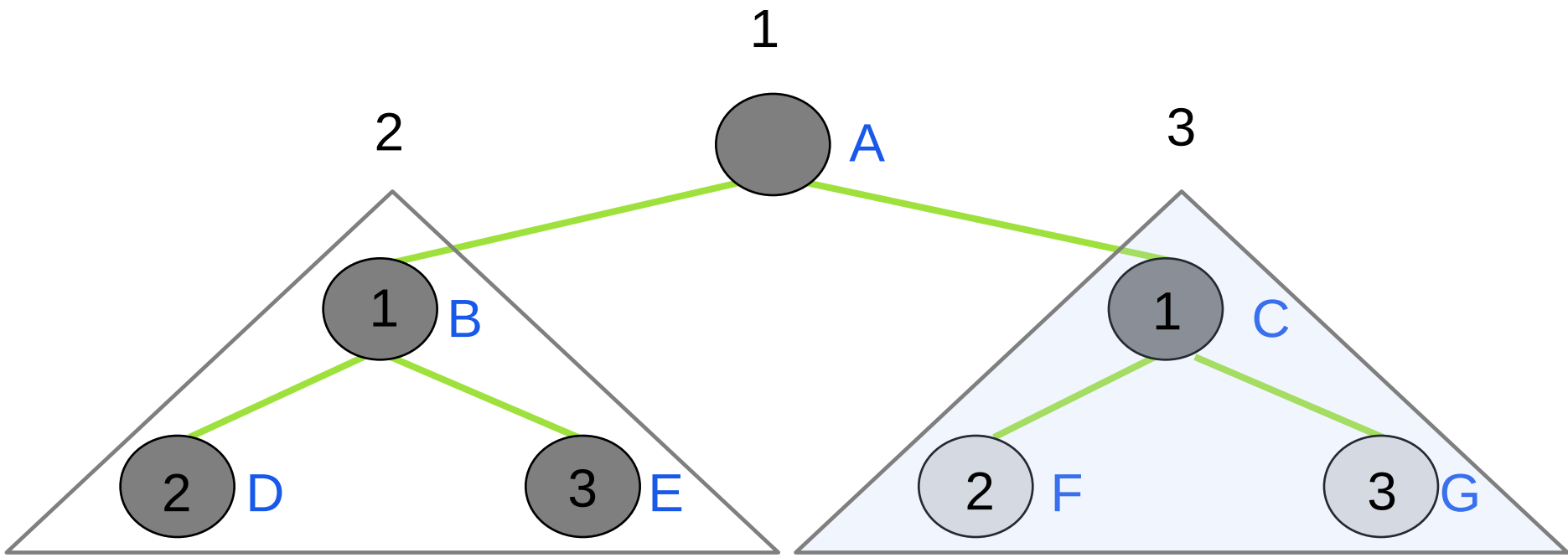
Output: A→B→D→E

Preorder Traversal



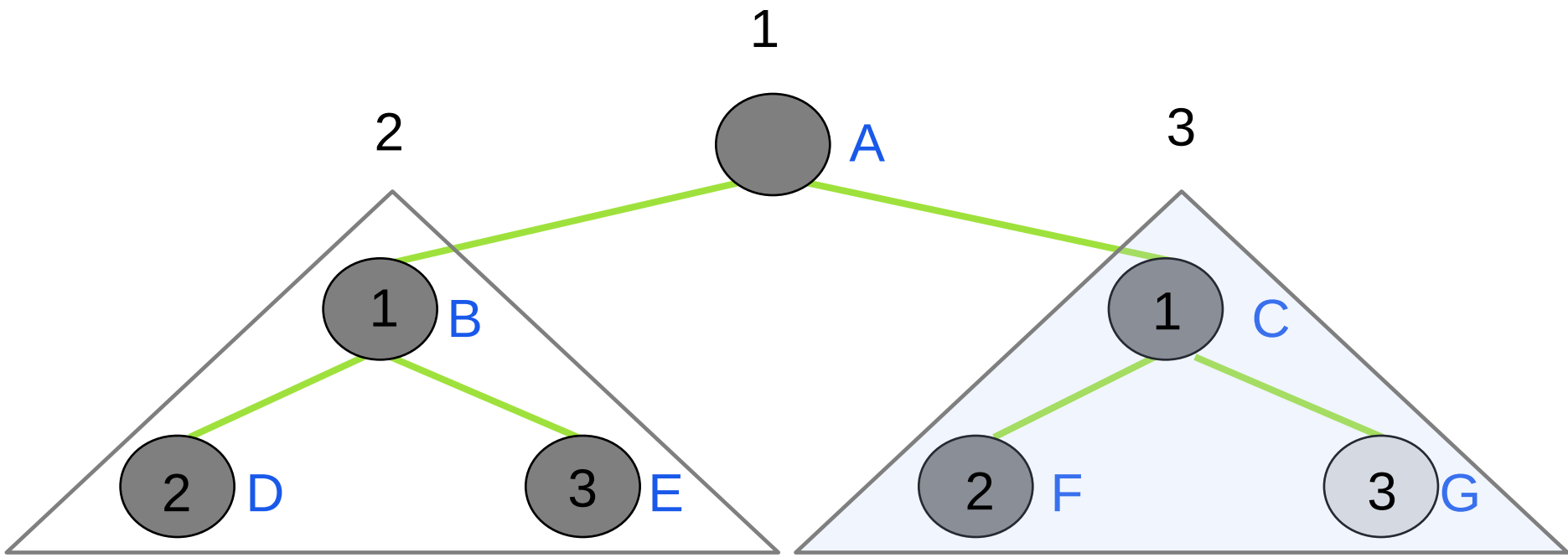
Output: A→B→D→E

Preorder Traversal



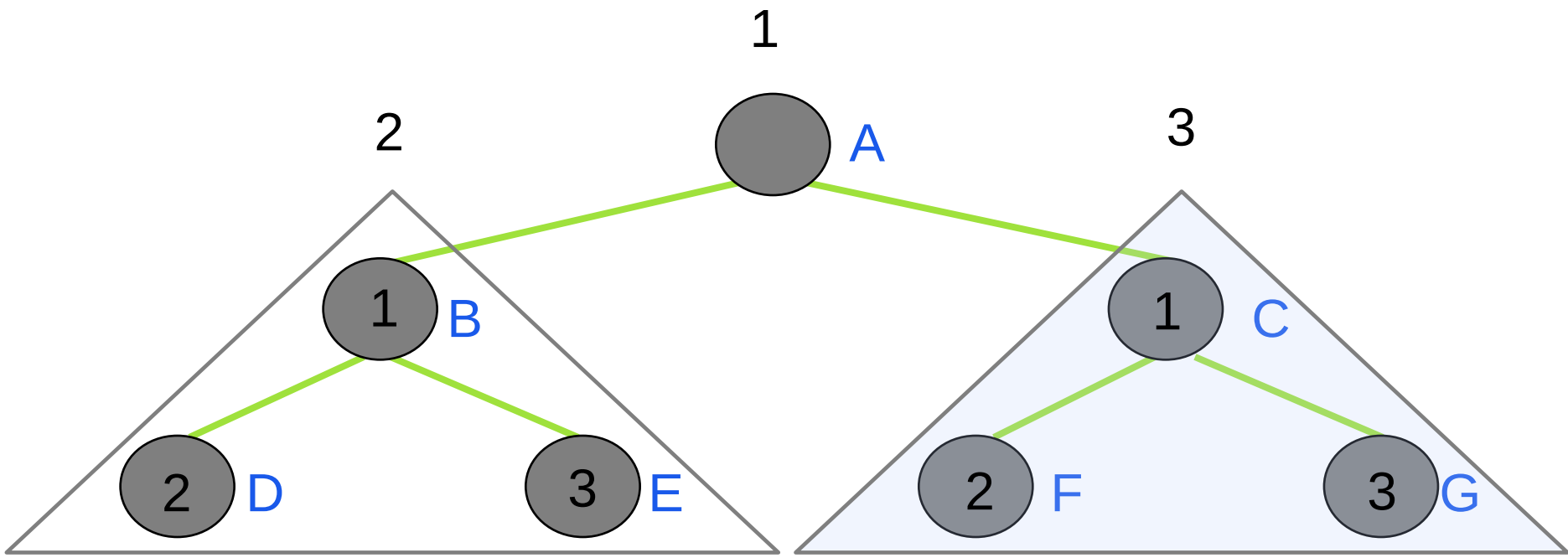
Output: A→B→D→E→C

Preorder Traversal



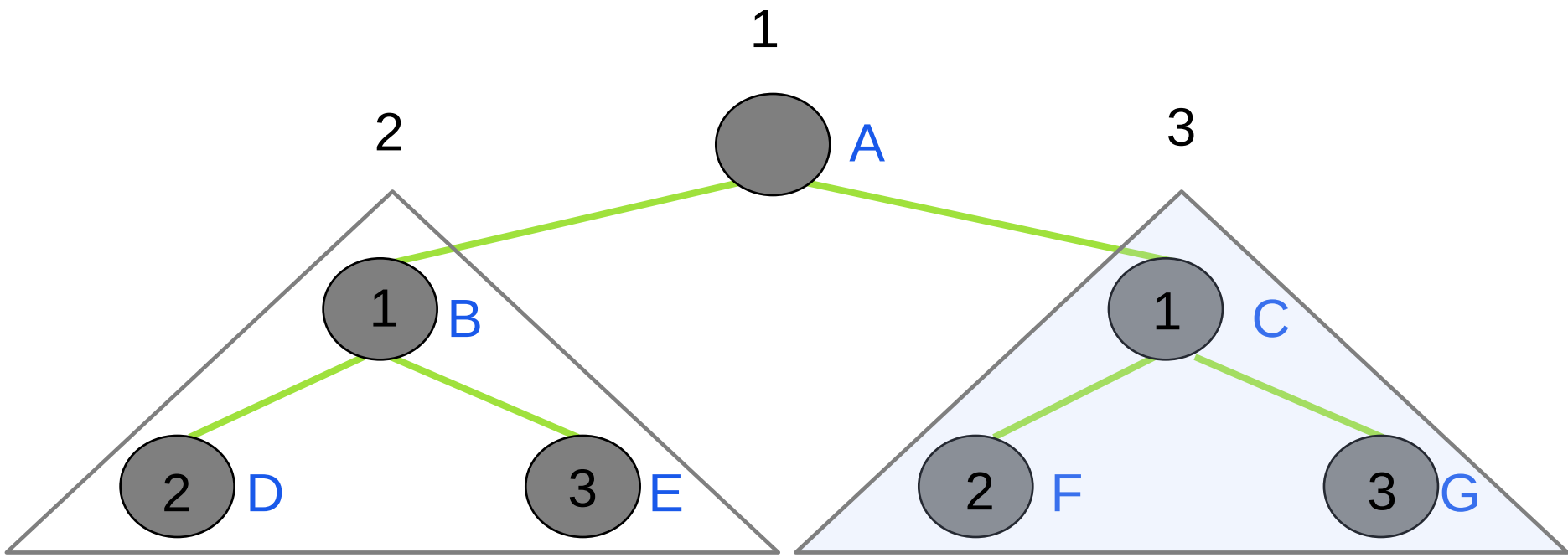
Output: A→B→D→E→C→F

Preorder Traversal



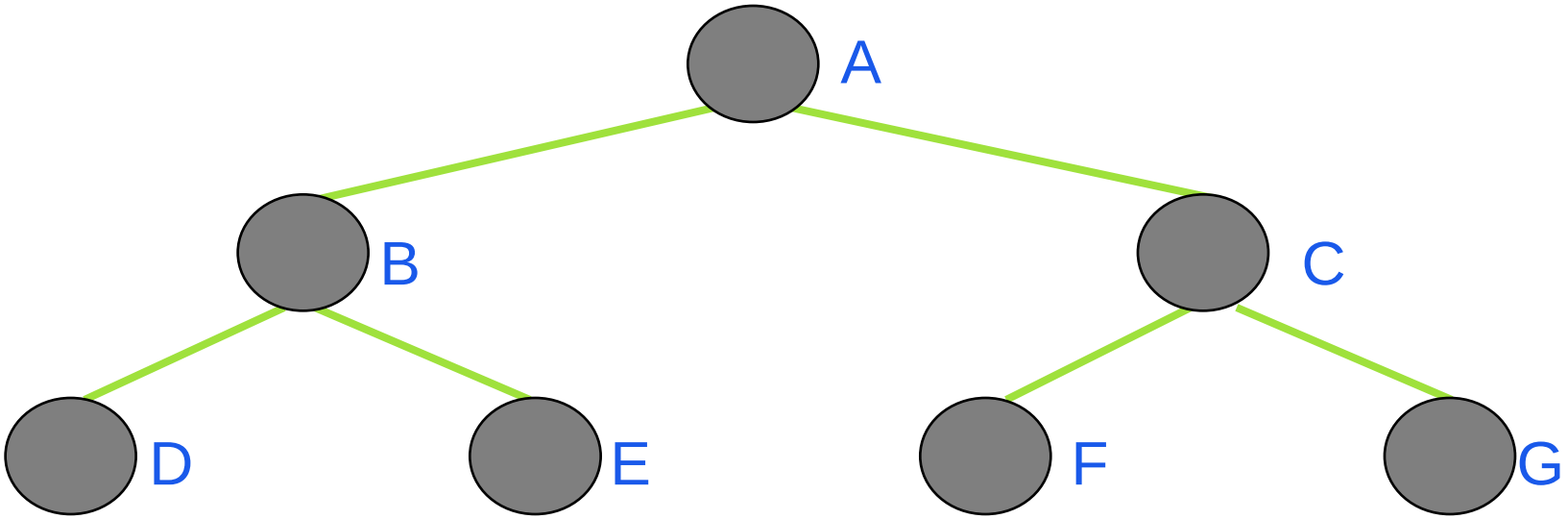
Output: A→B→D→E→C→F

Preorder Traversal



Output: A→B→D→E→C→F→G

Preorder Traversal



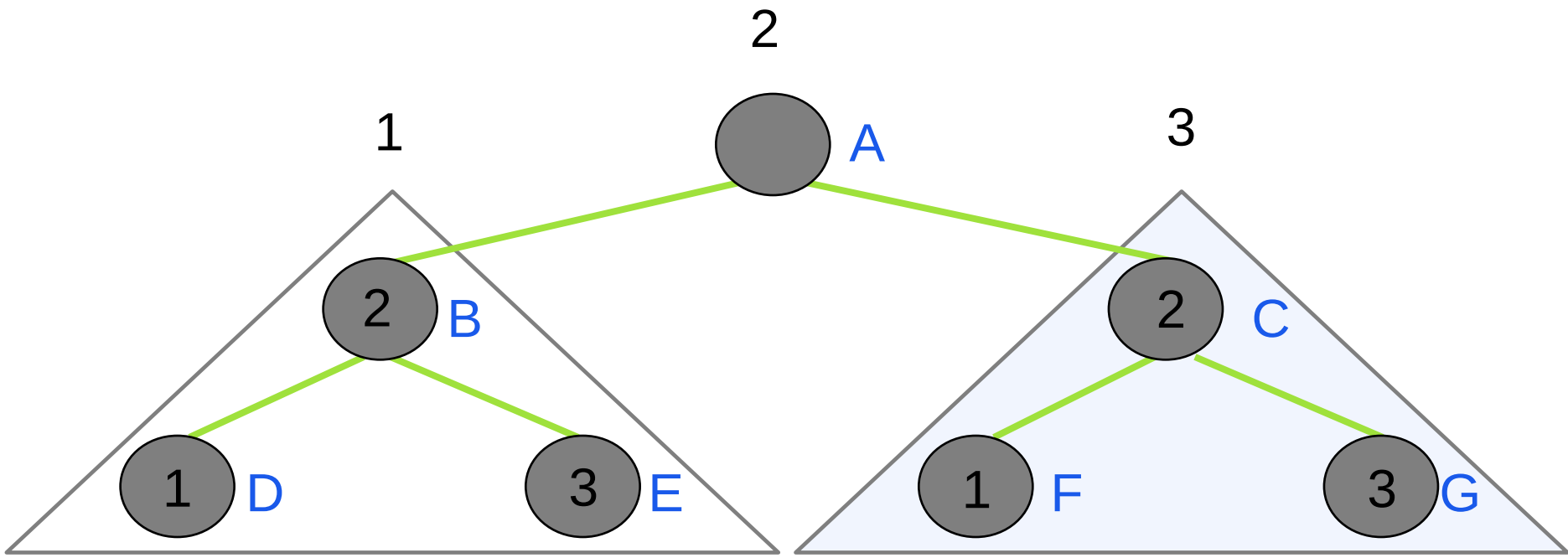
Output: A→B→D→E→C→F→G

Inorder Traversal

- Start at the root
- Visit the left child of each node, then the node, then any remaining nodes

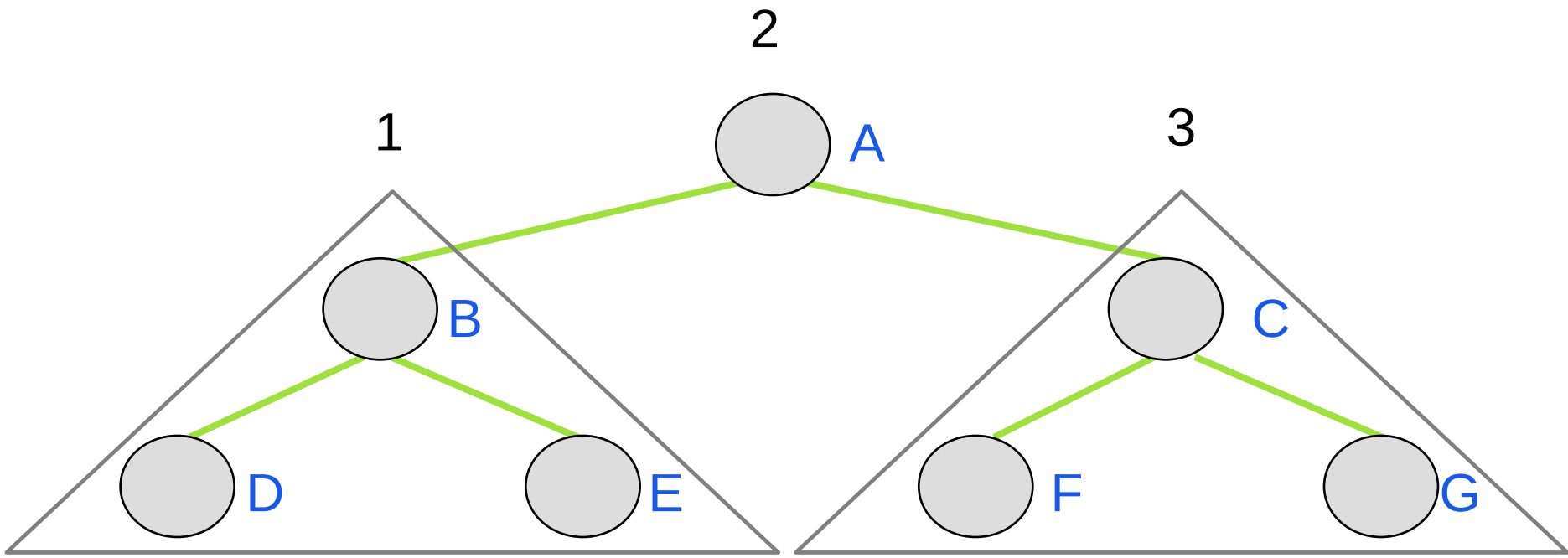
- Recursive algorithm for inorder traversal
  - If tree is not empty,
    Recursively traverse left subtree
    Visit root node of tree
    Recursively traverse right subtree

Inorder

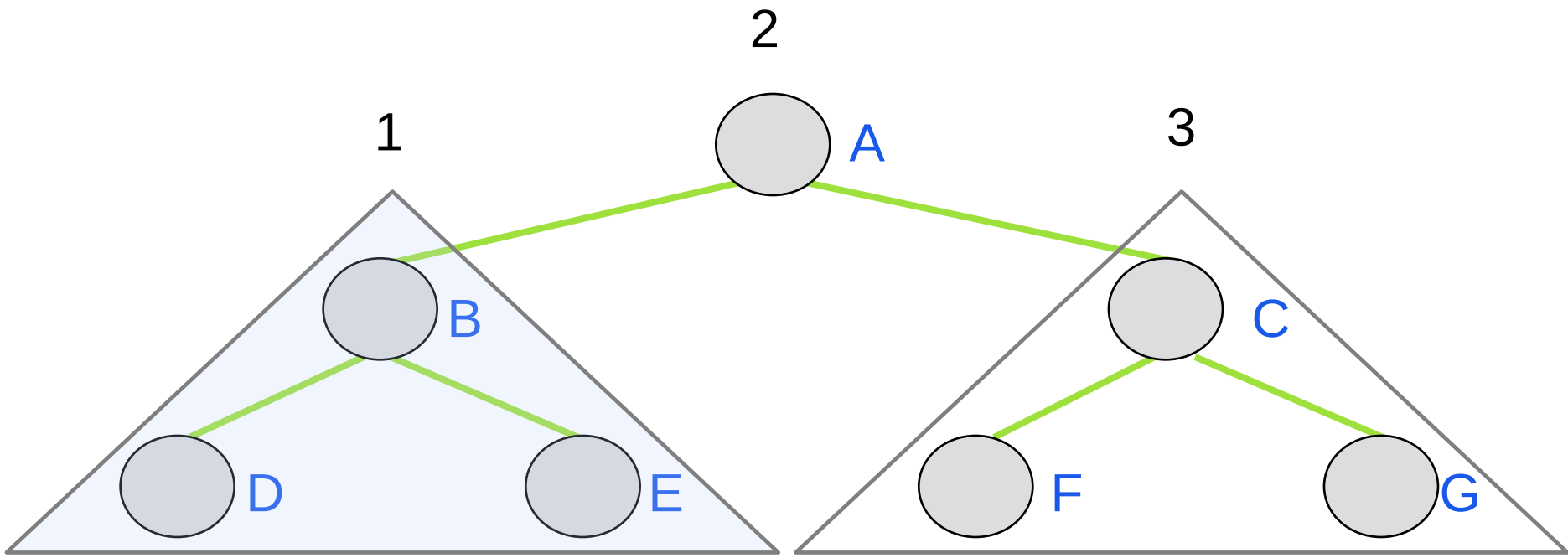


Output: D→B→E→A→F→C→G

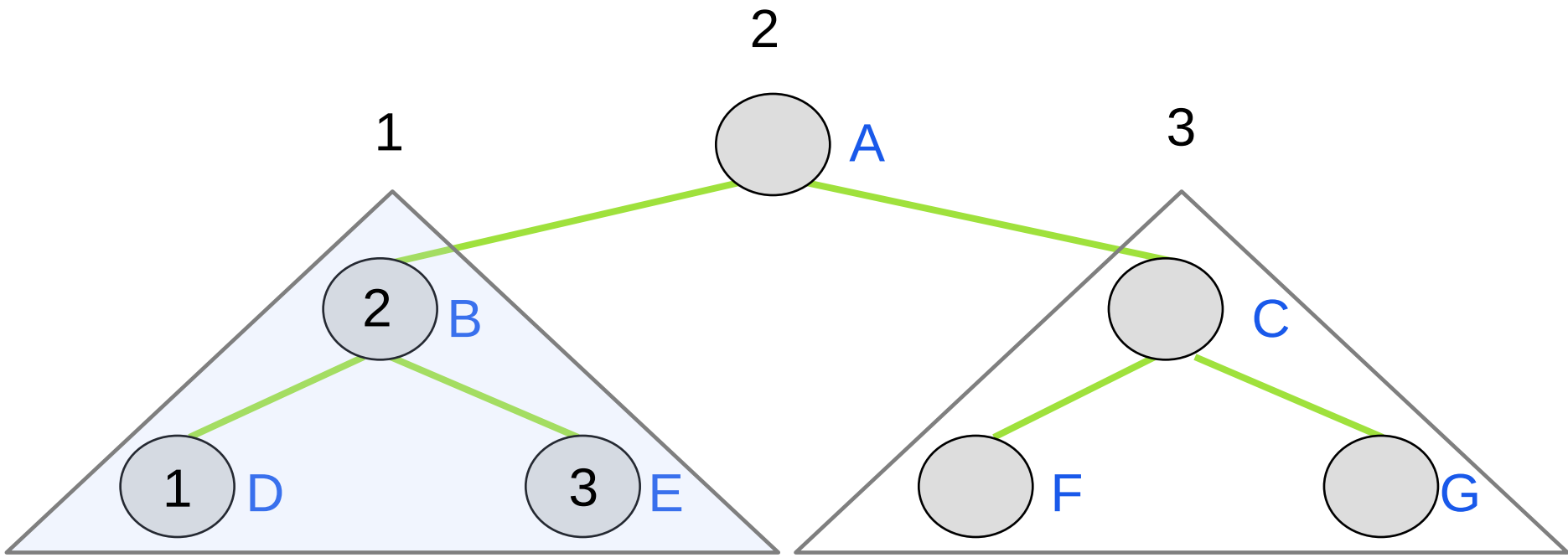
Inorder



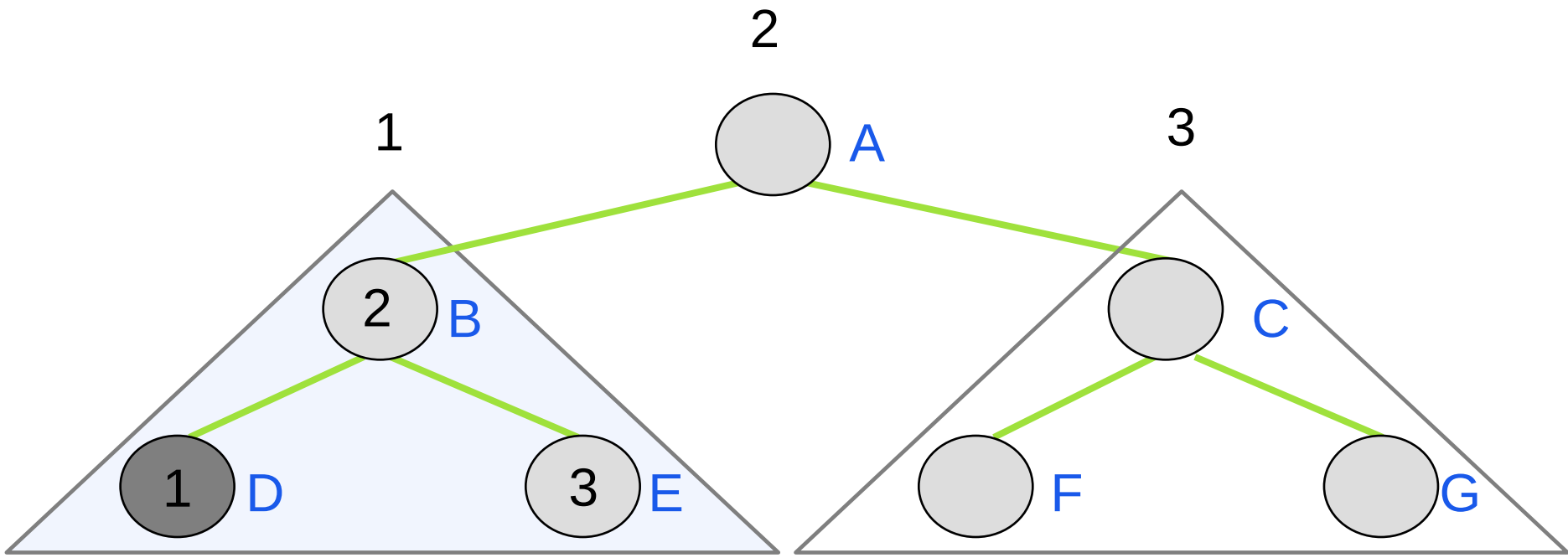
Inorder



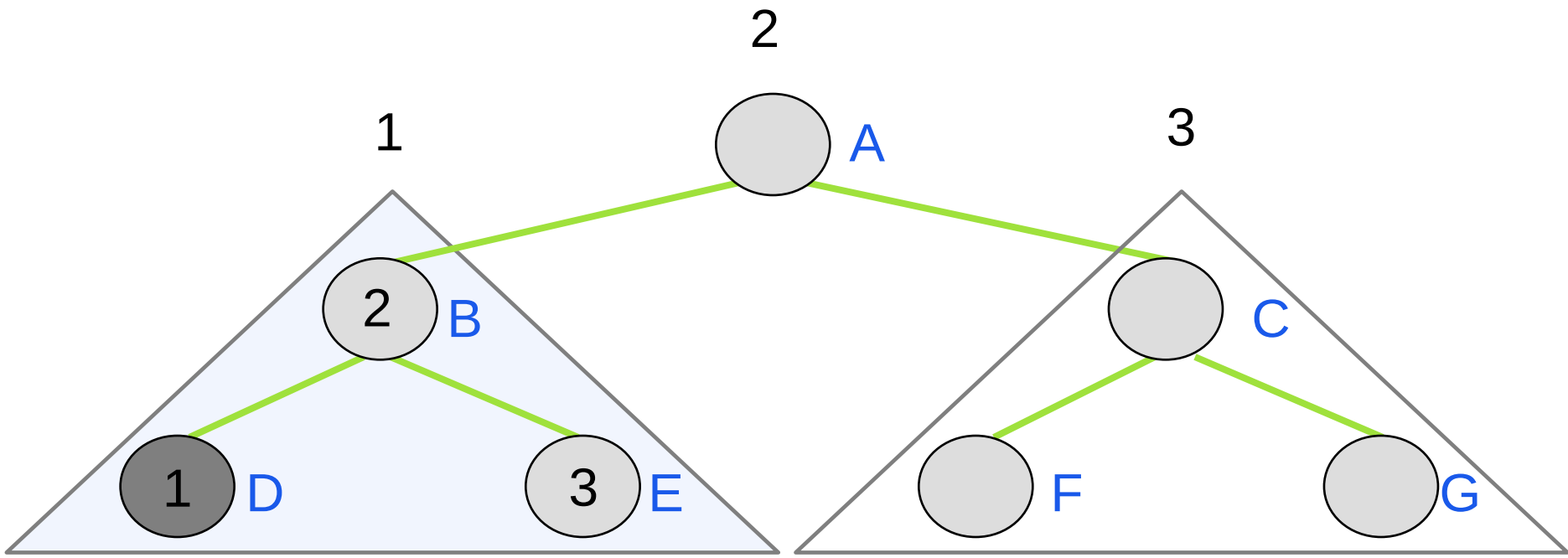
Inorder



Inorder

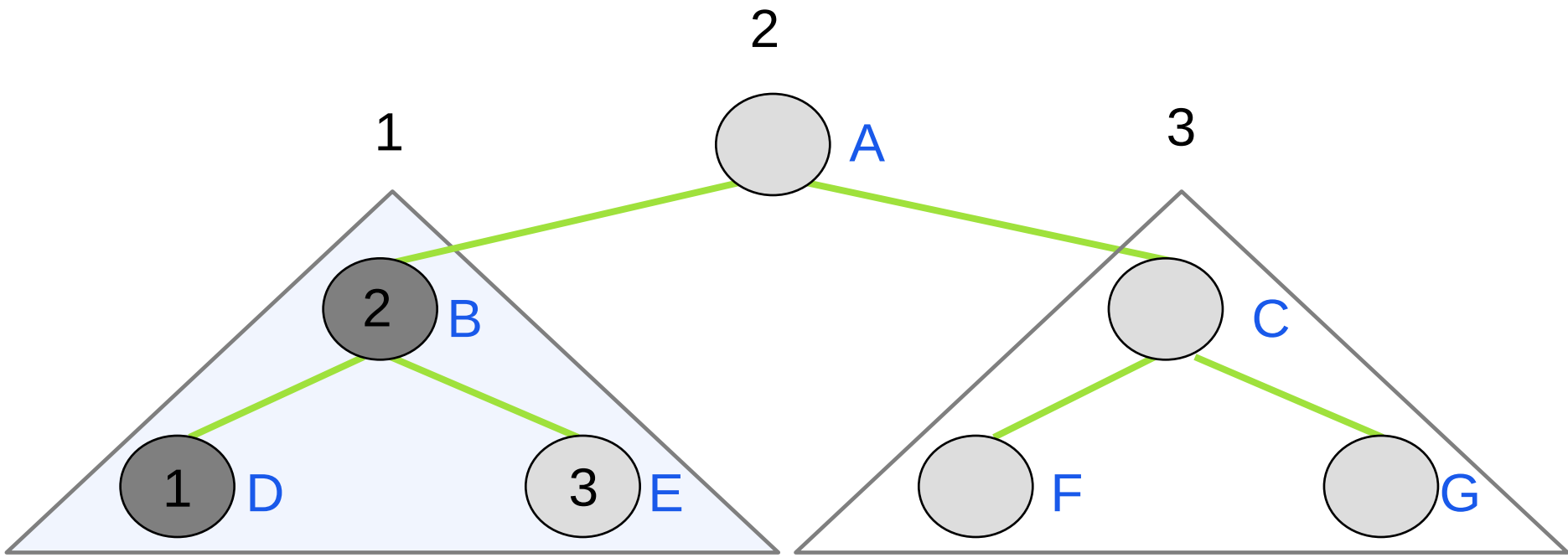


Inorder



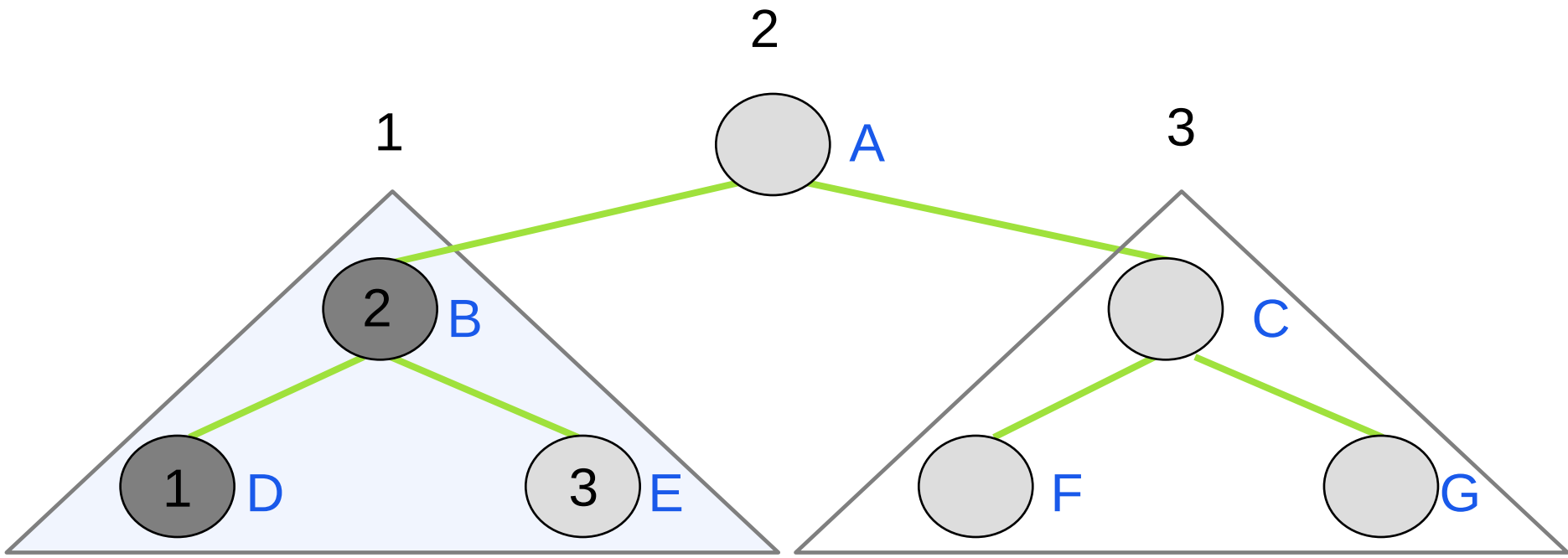
Output: D

Inorder



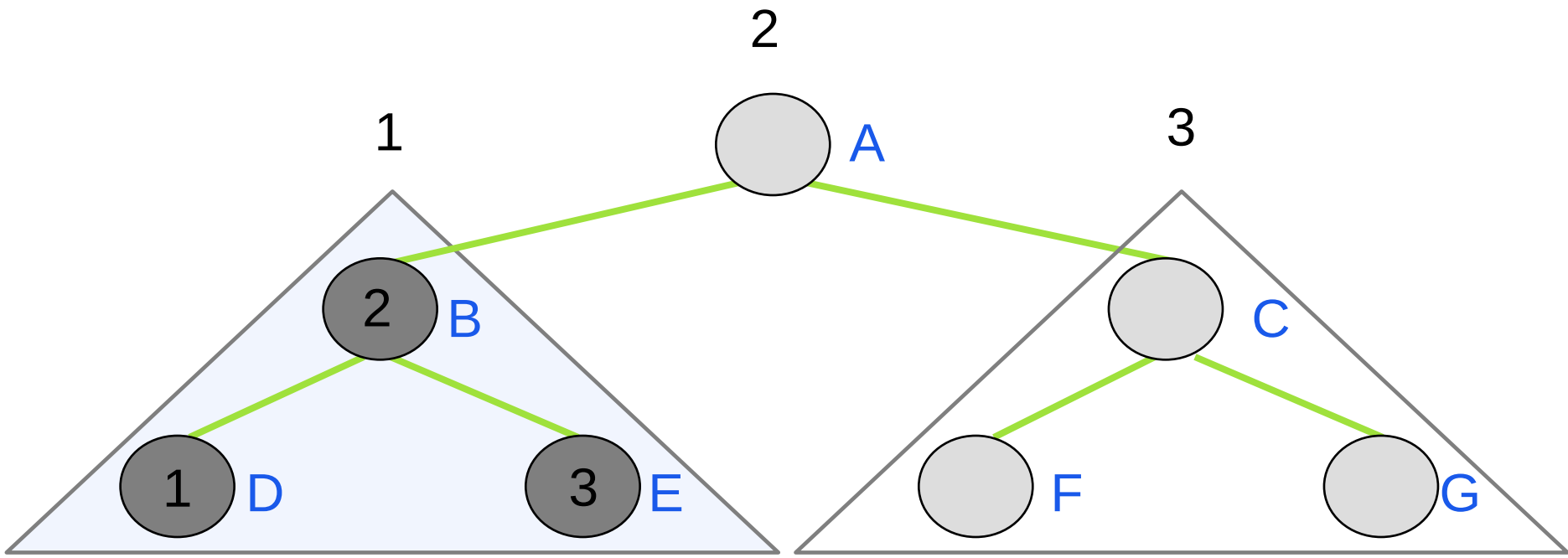
Output: D

Inorder



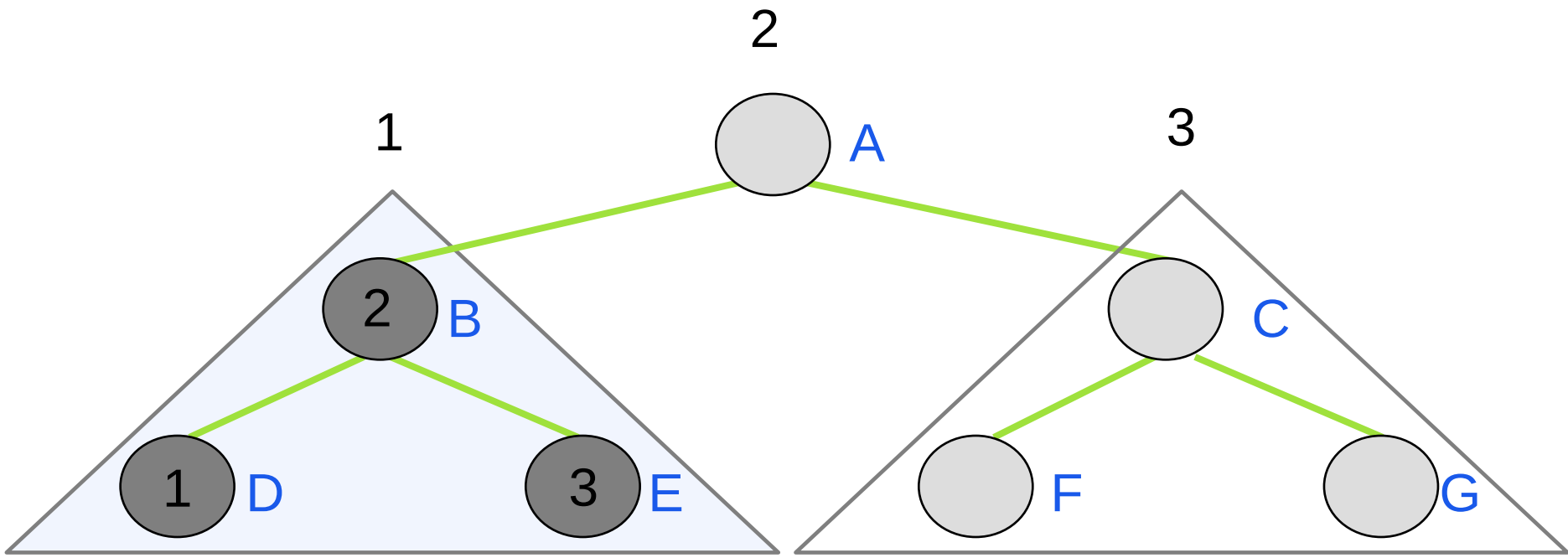
Output: D→B

Inorder



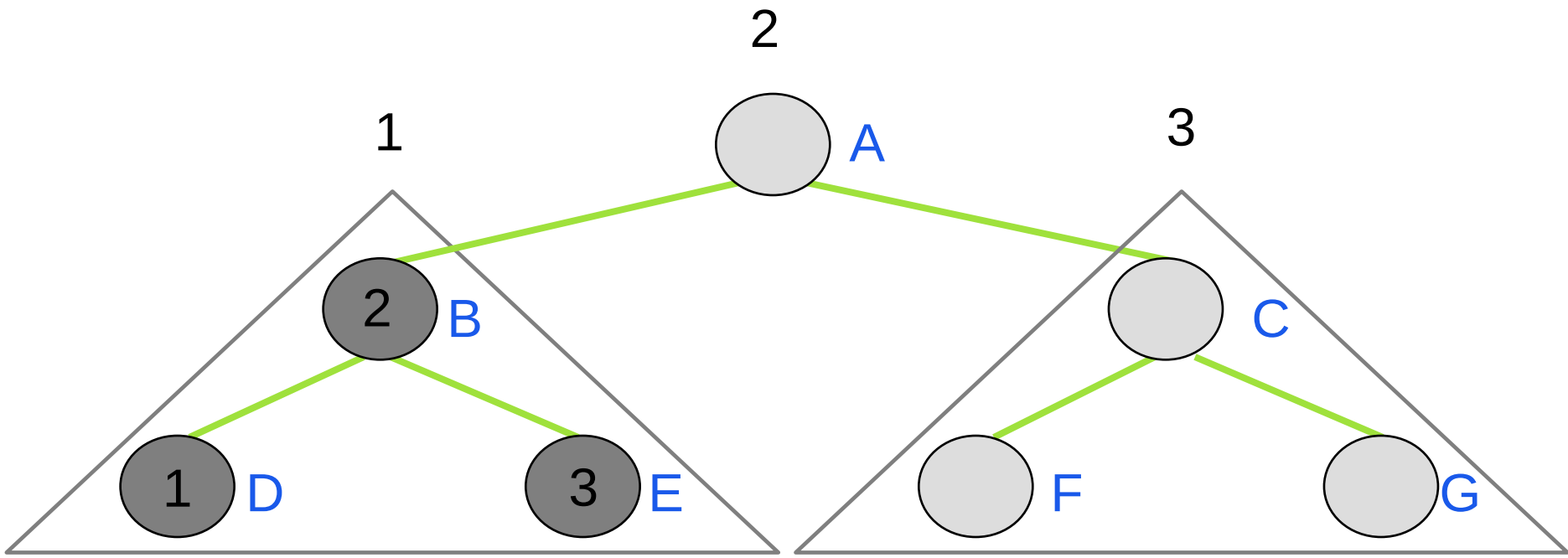
Output: D→B

Inorder



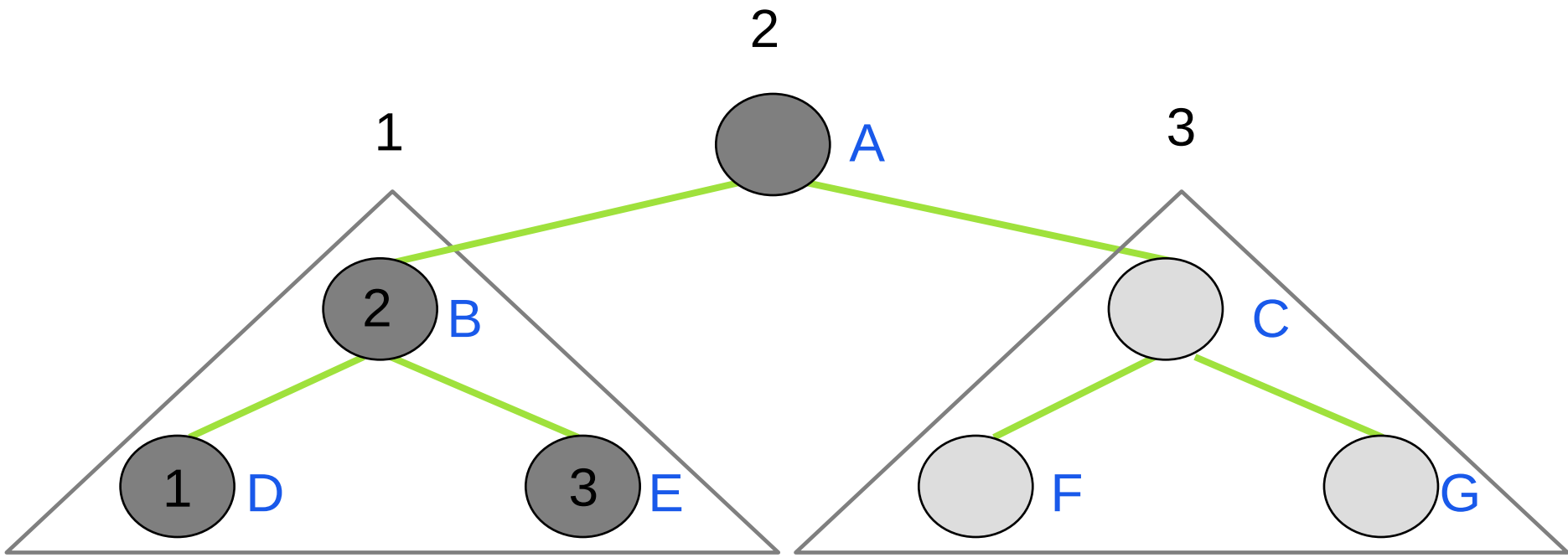
Output: D→B→E

Inorder



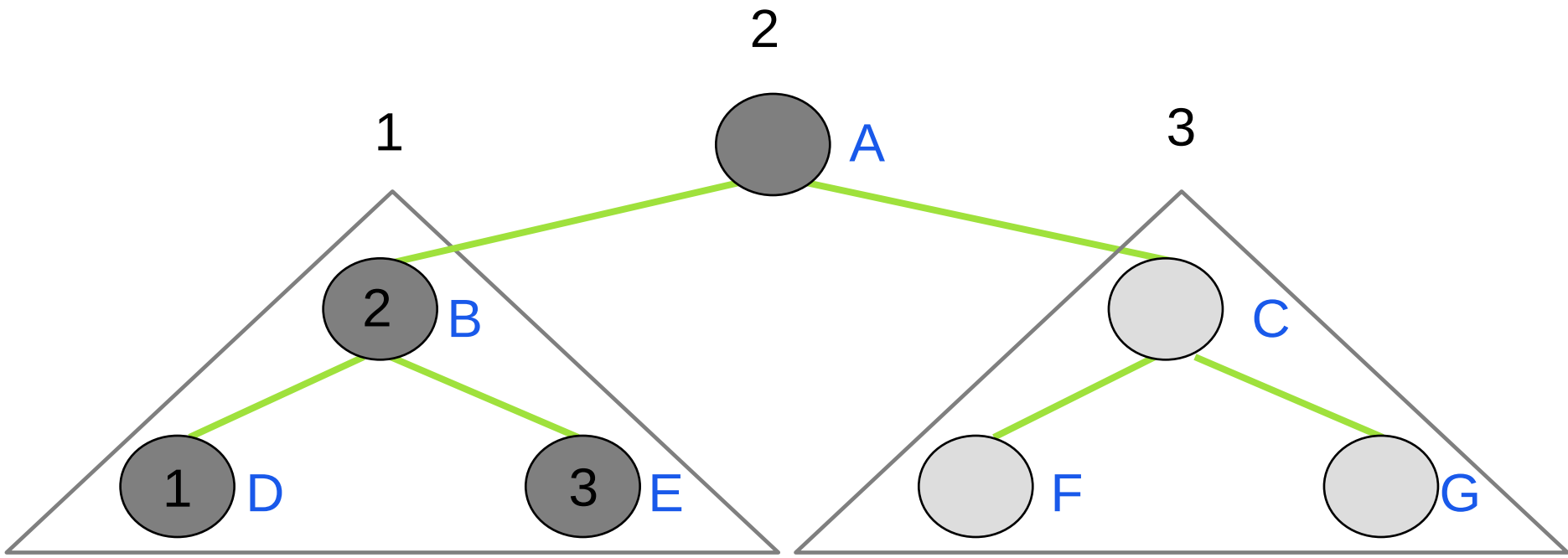
Output: D→B→E

Inorder



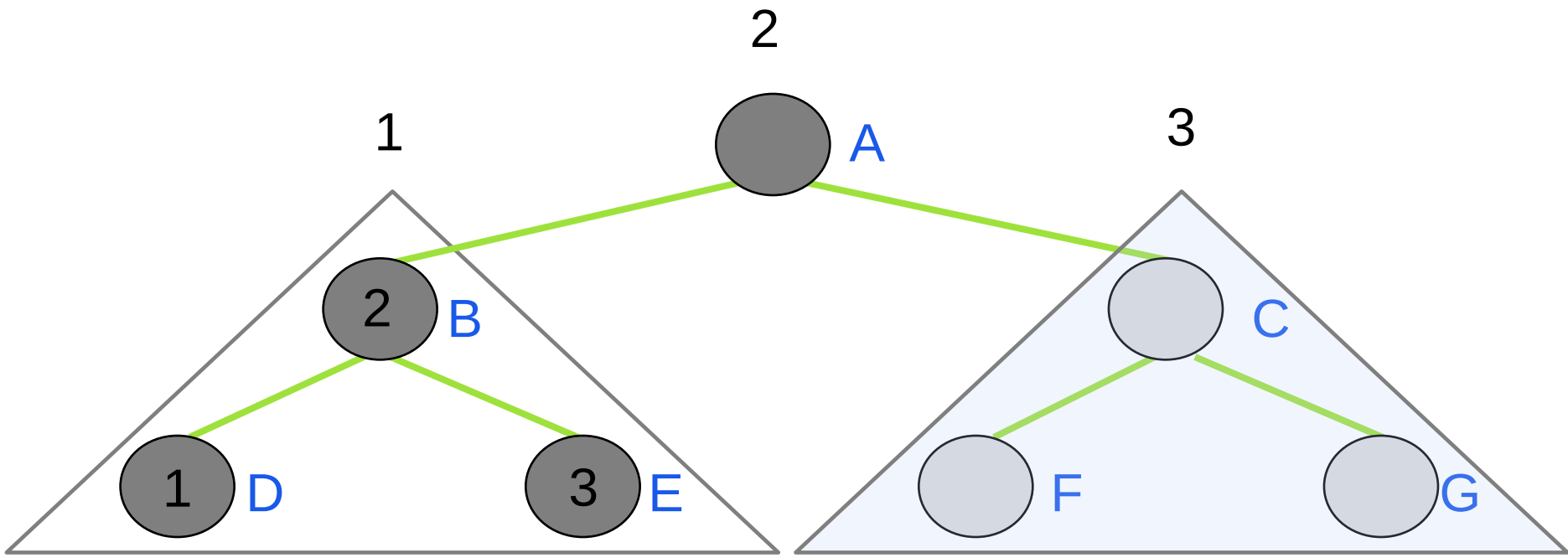
Output: D→B→E

Inorder



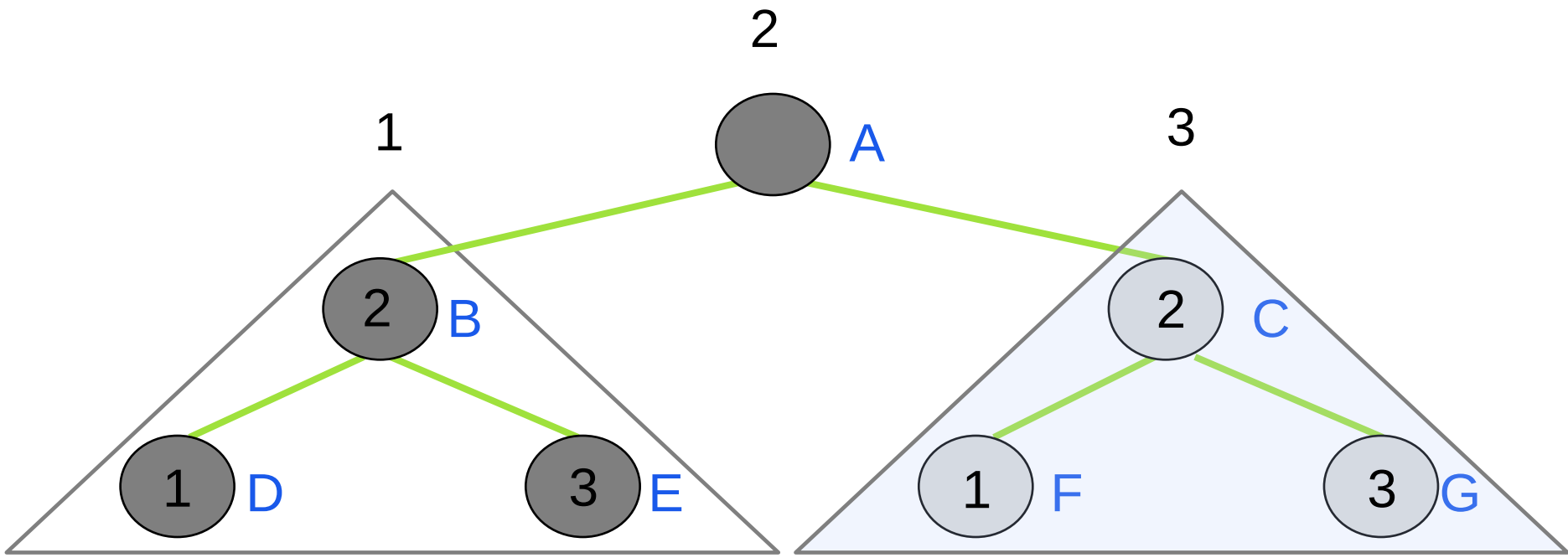
Output: D→B→E→A

Inorder



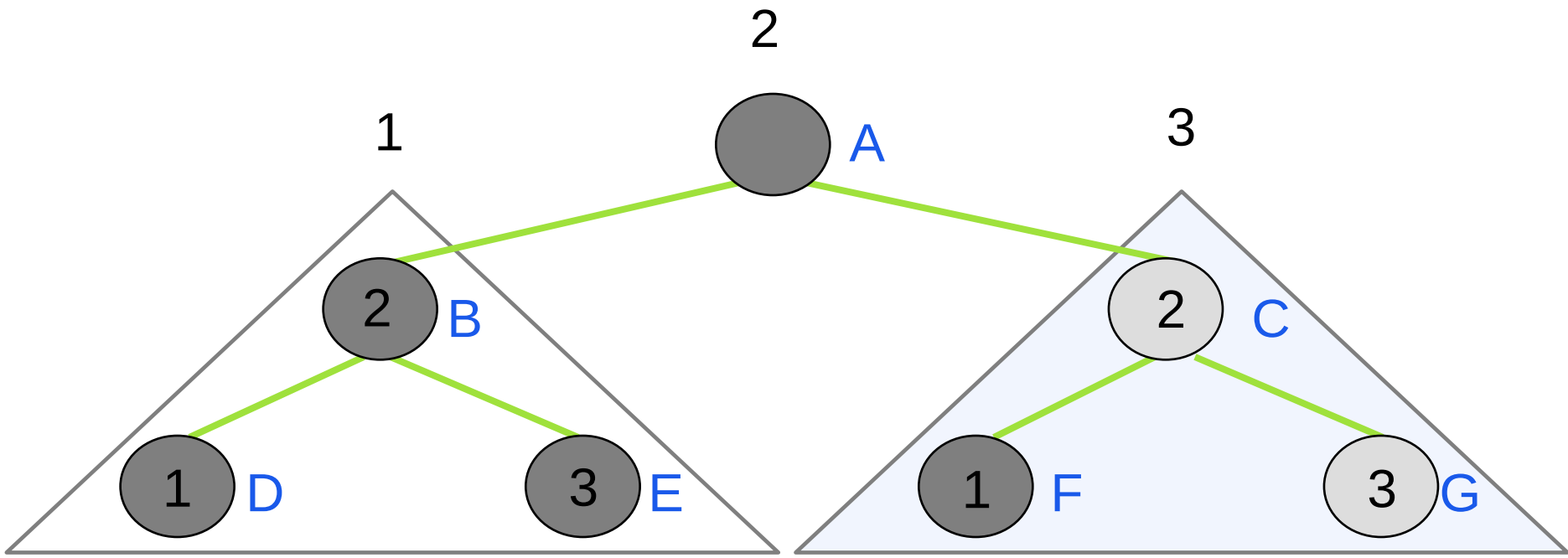
Output: D→B→E→A

Inorder



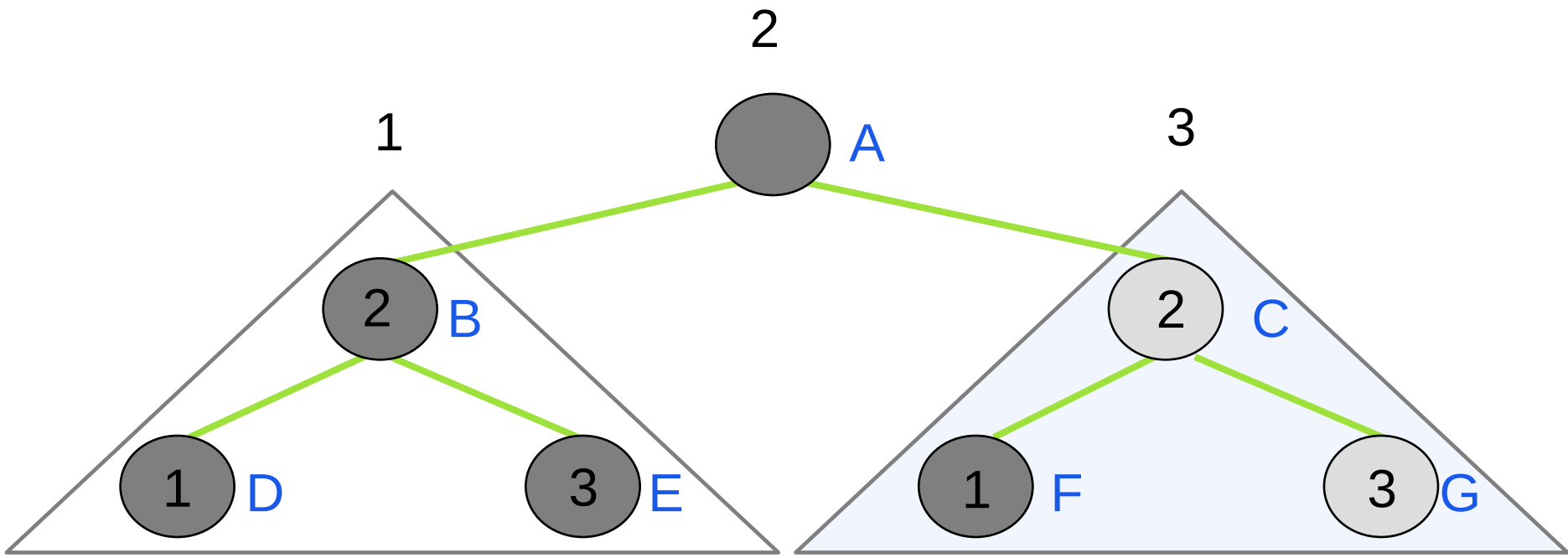
Output: D→B→E→A

Inorder



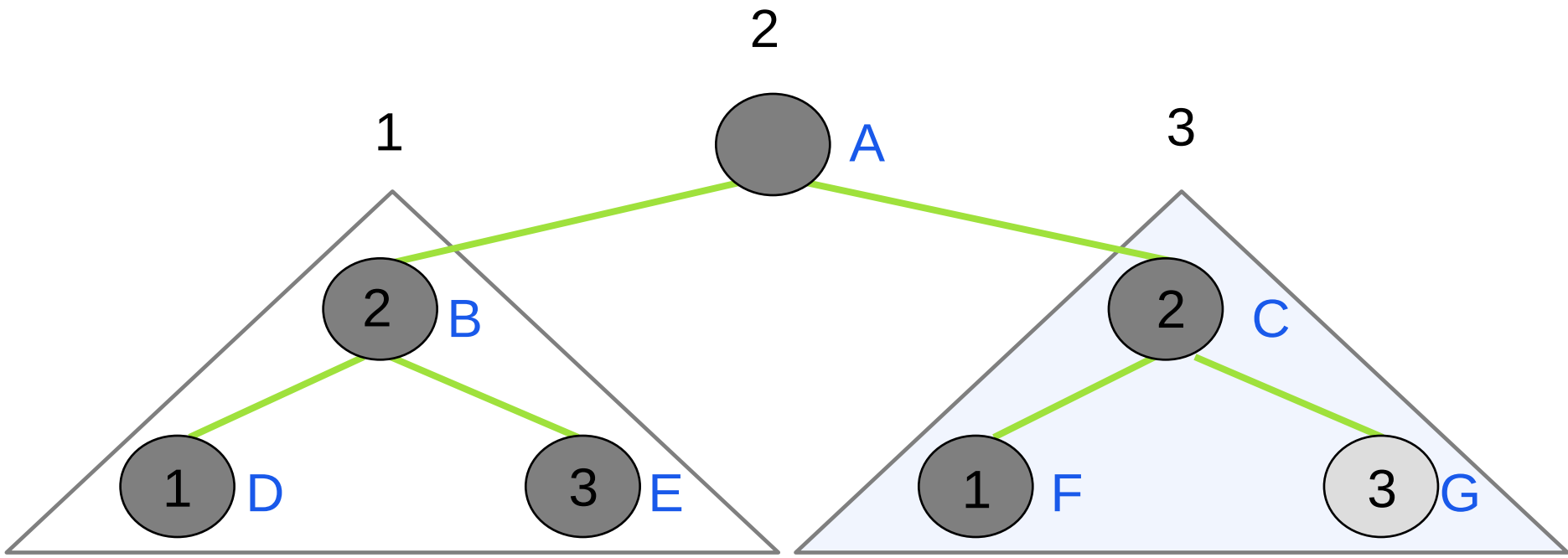
Output: D→B→E→A

Inorder



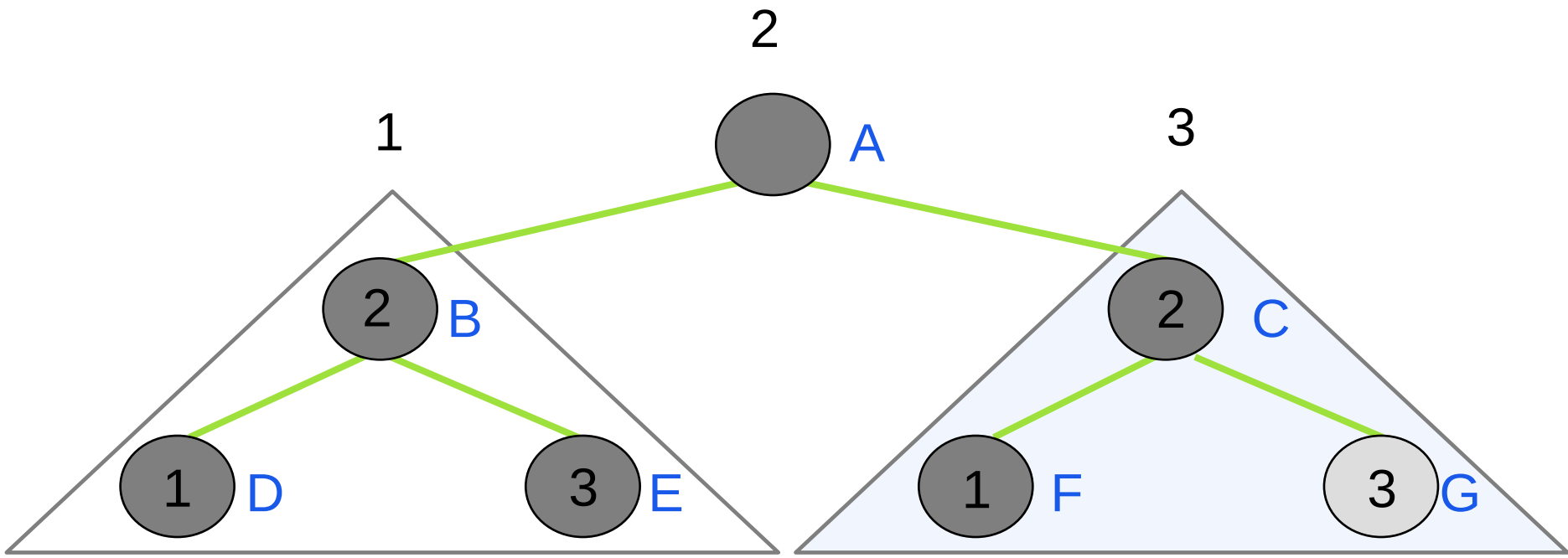
Output: D→B→E→A→F

Inorder



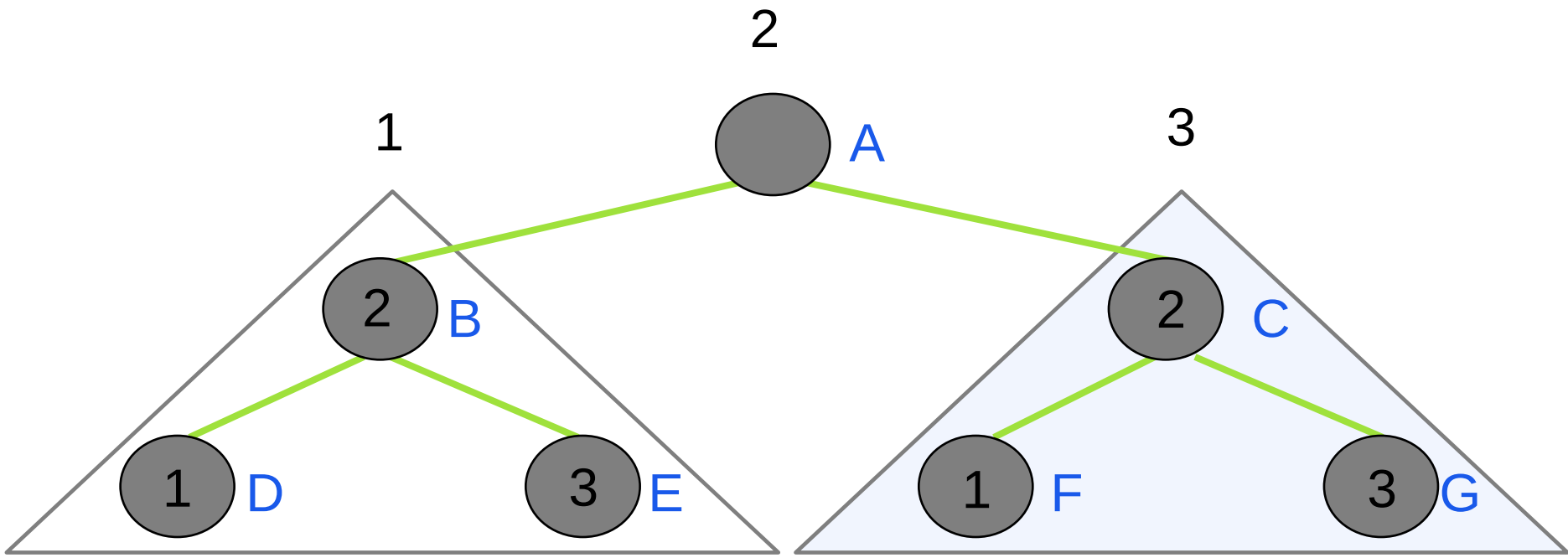
Output: D→B→E→A→F

Inorder



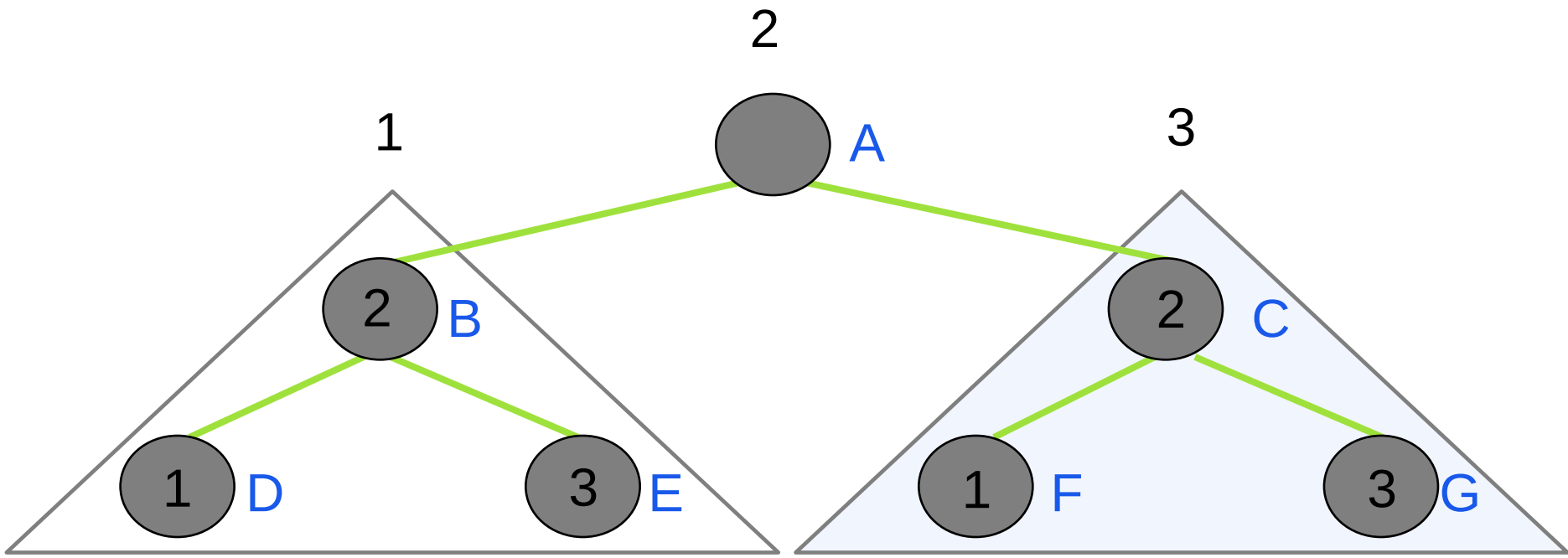
Output: D→B→E→A→F→C

Inorder



Output: D→B→E→A→F→C

Inorder



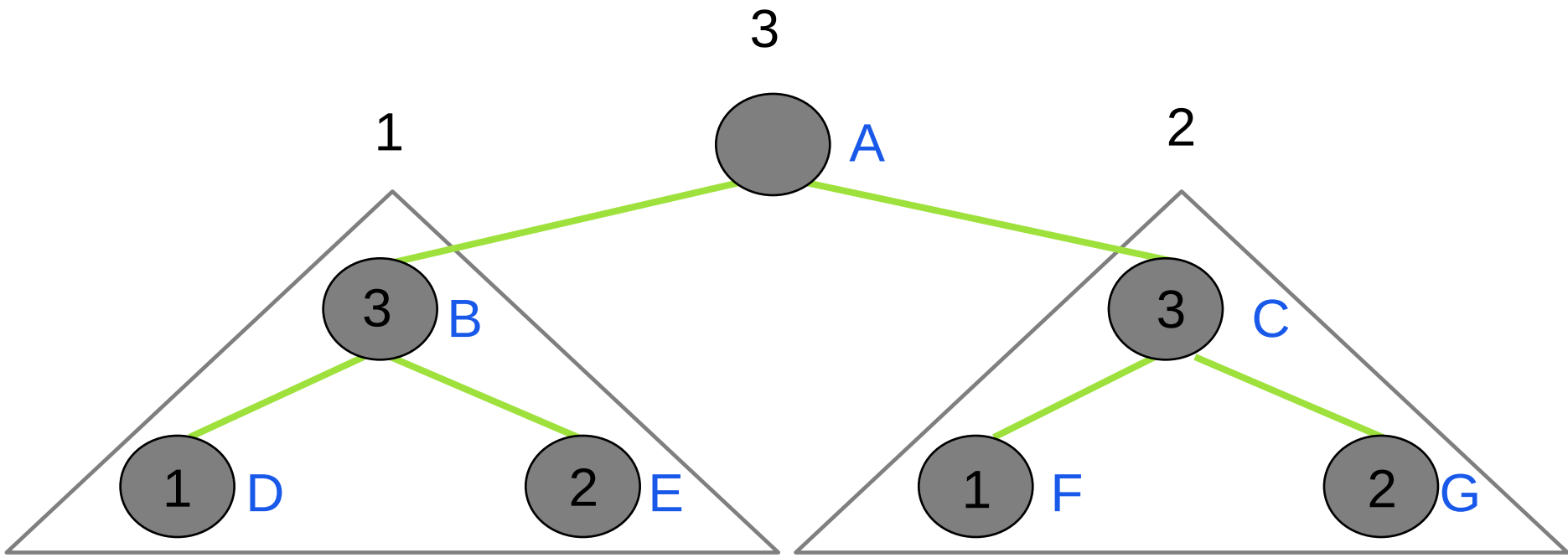
Output: D→B→E→A→F→C→G

Postorder Traversal

- Start at the root
- Visit the children of each node, then the node

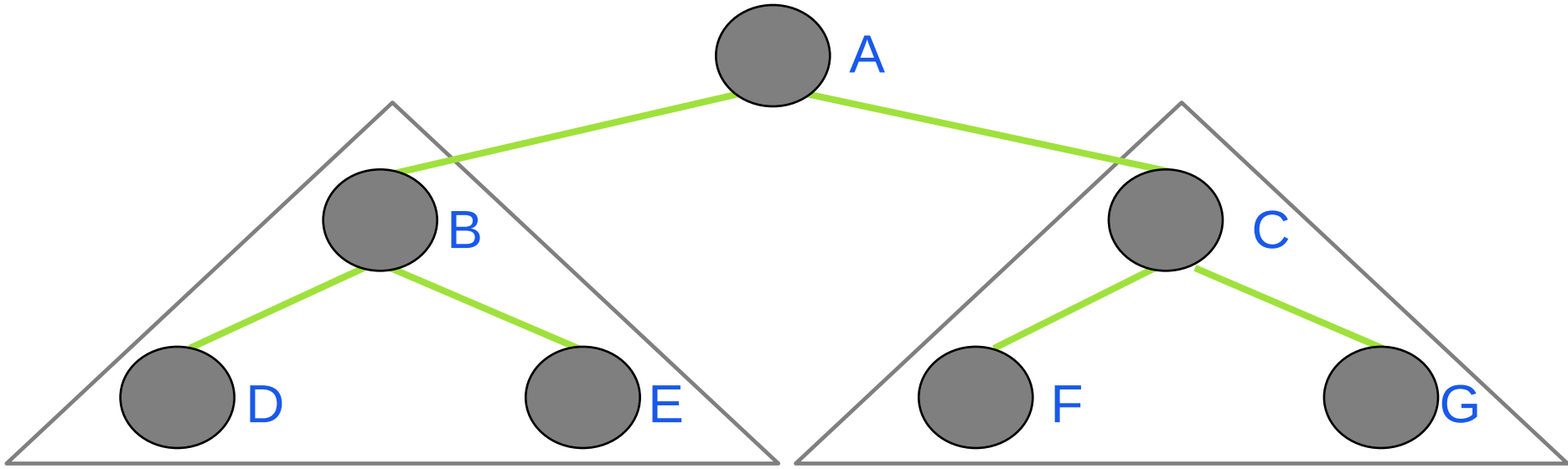
- Recursive algorithm for postorder traversal
  - If tree is not empty,
    Recursively traverse left subtree
    Recursively traverse right subtree
    Visit root node of tree

Postorder

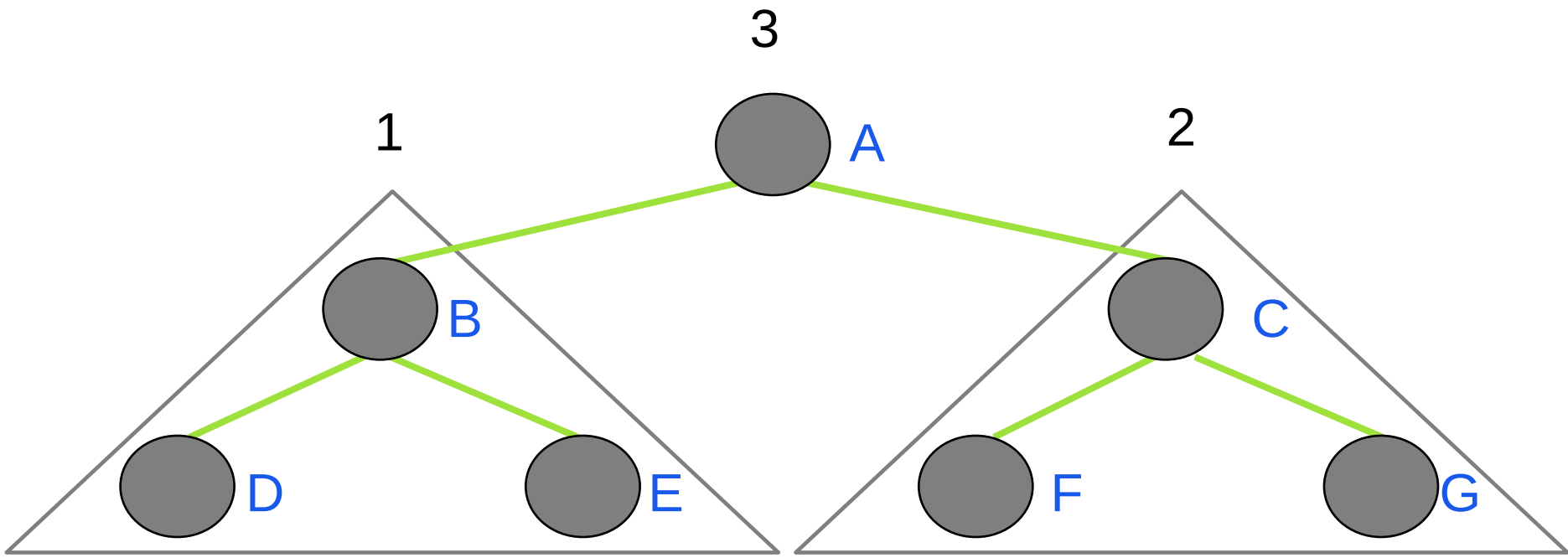


Output: D→E→B→F→G → C→A

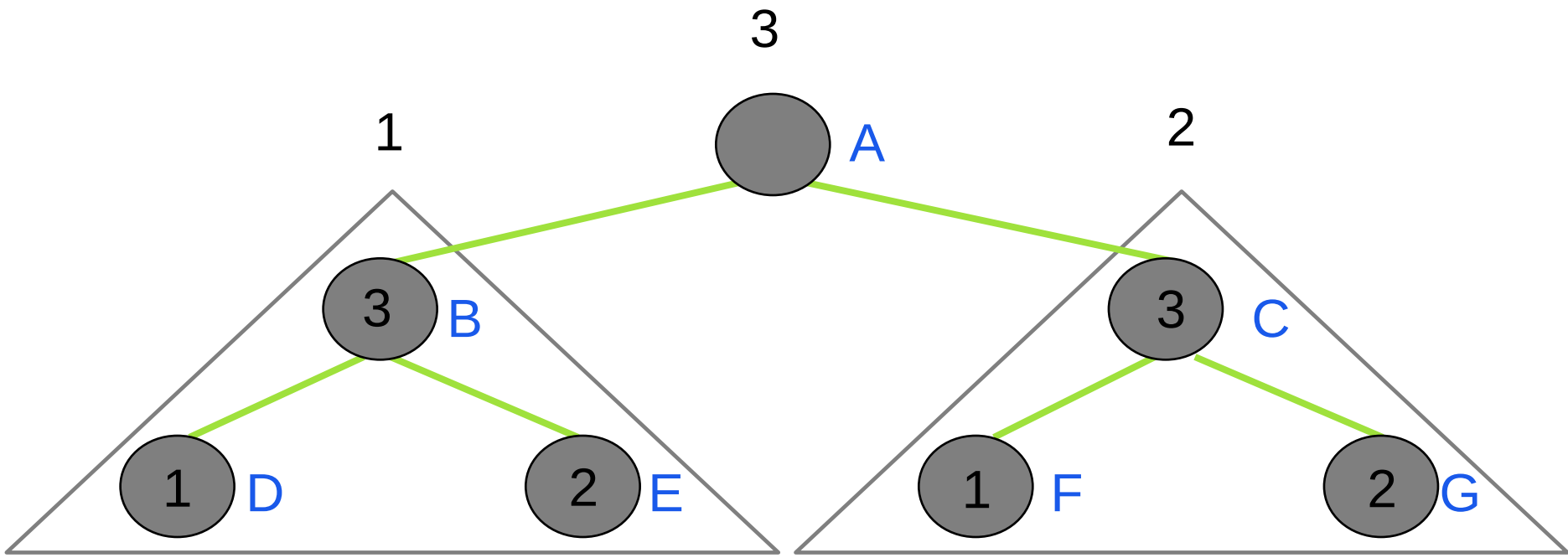
Postorder



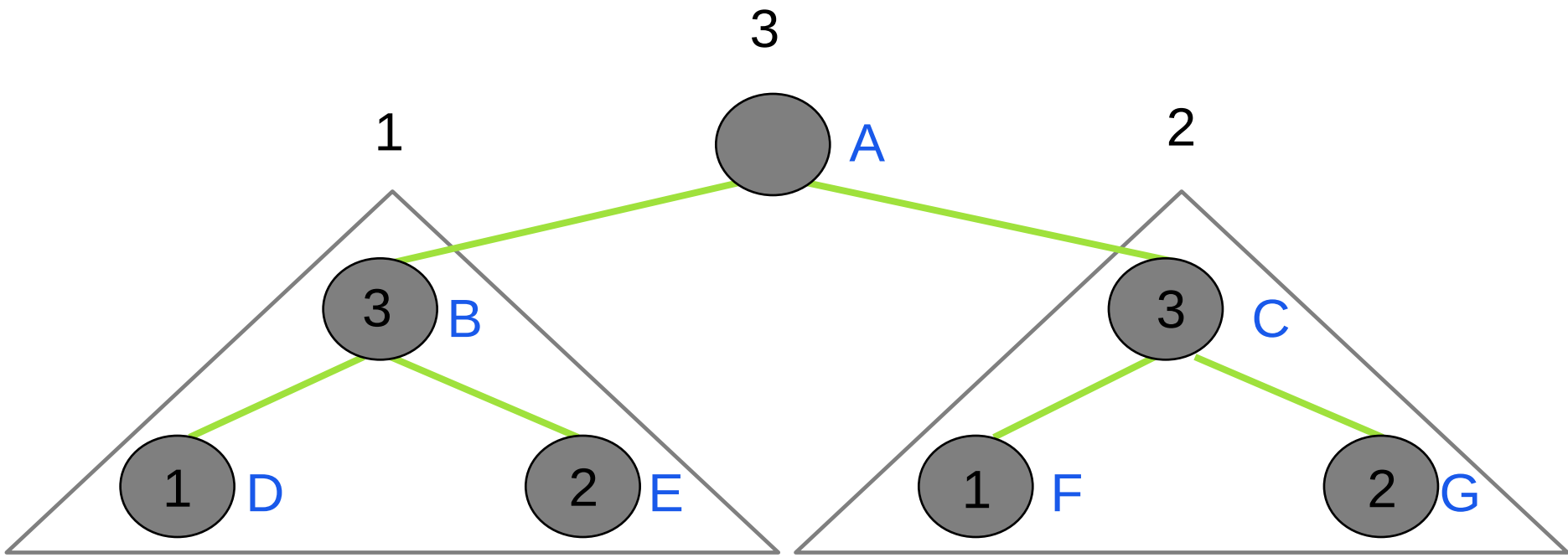
Postorder



Postorder



Postorder



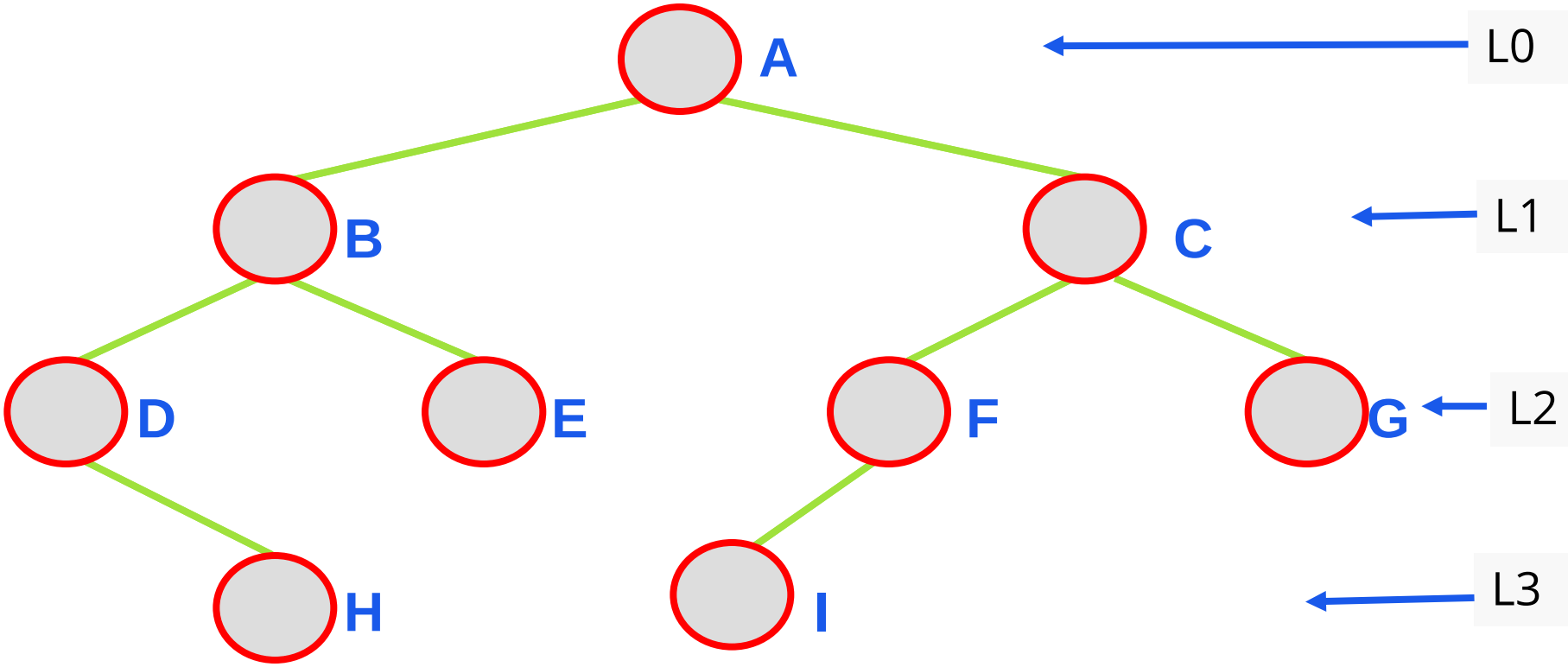
Output: D→E→B→F→G → C→A

Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right

Level Order Traversal

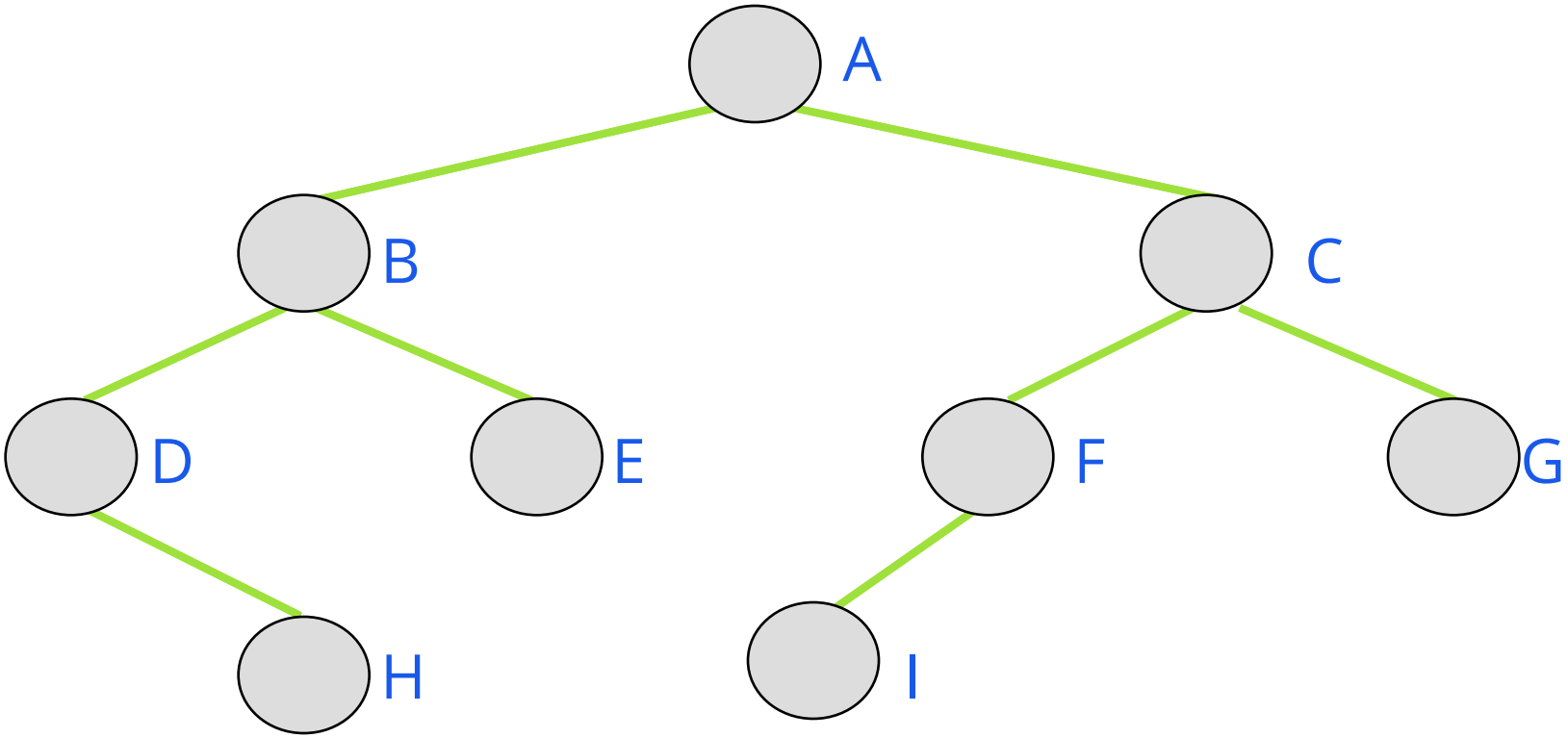
- Start at the root
- Visit the nodes at each level, from left to right



Output: A B C D E F G H I

Level Order Traversal

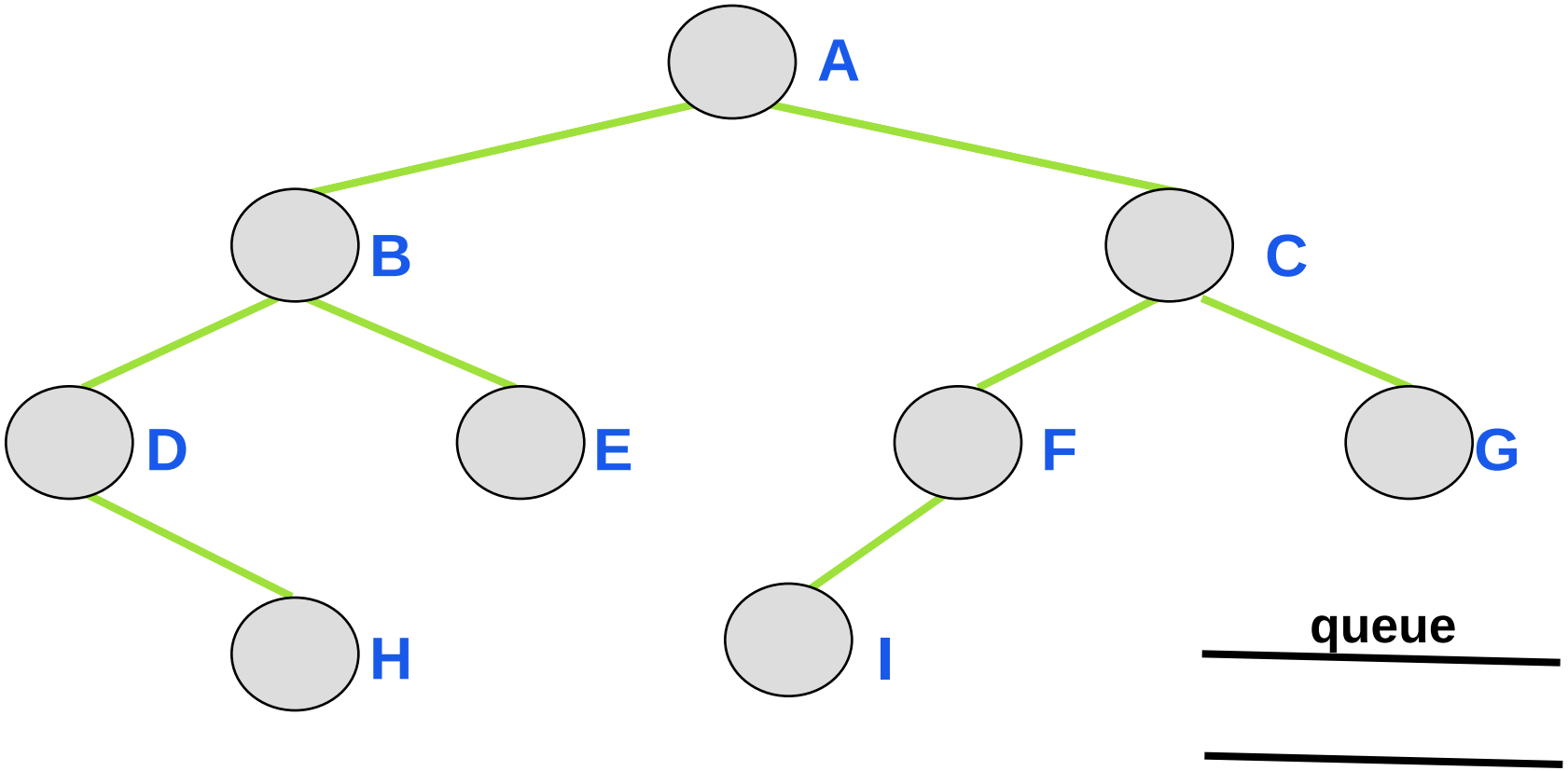
- Is there a recursive solution for this?
- What data structure can be used instead?



Nodes will be visited in the order ABCDEFGHI

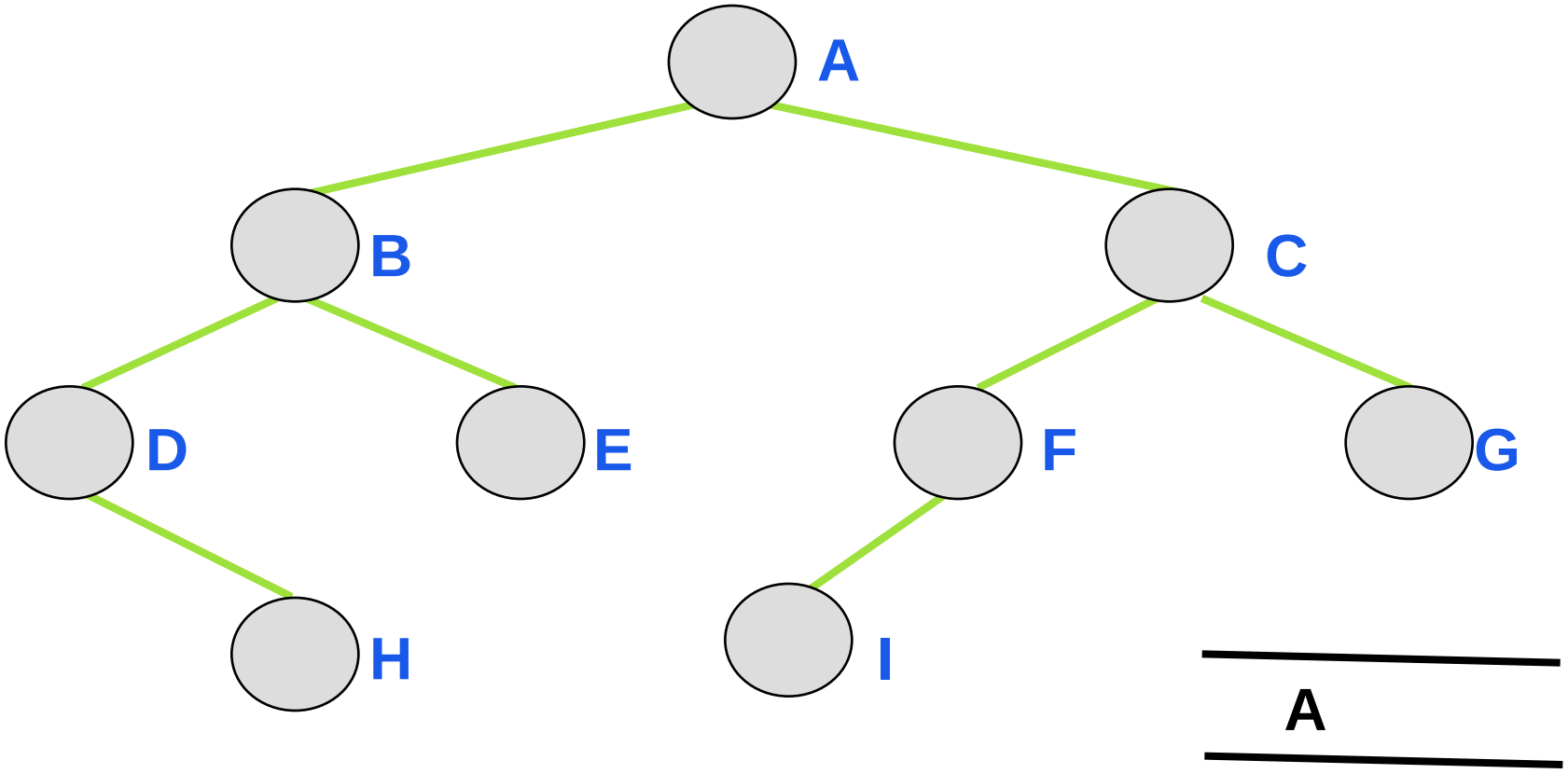
Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



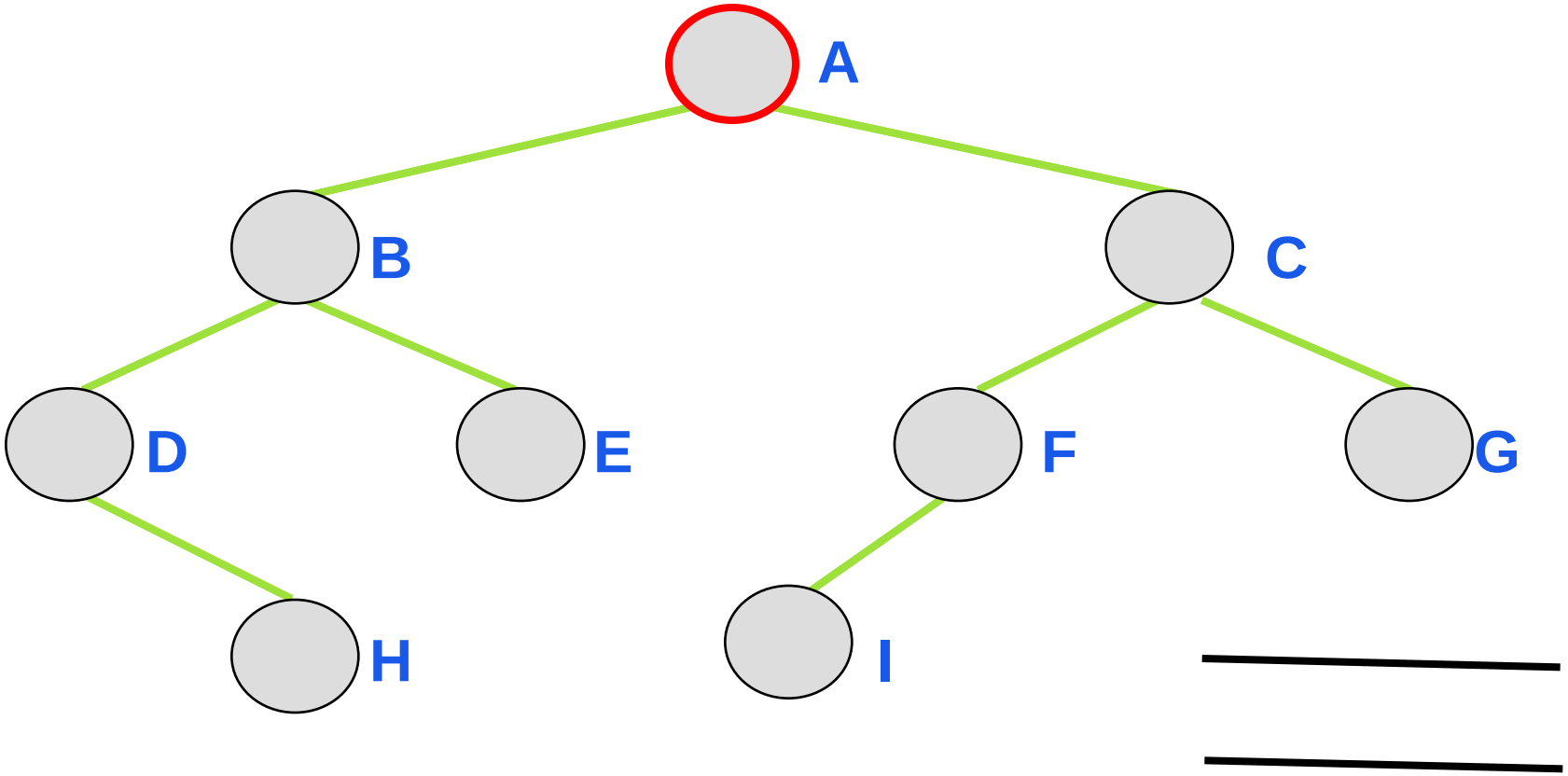
Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



Level Order Traversal

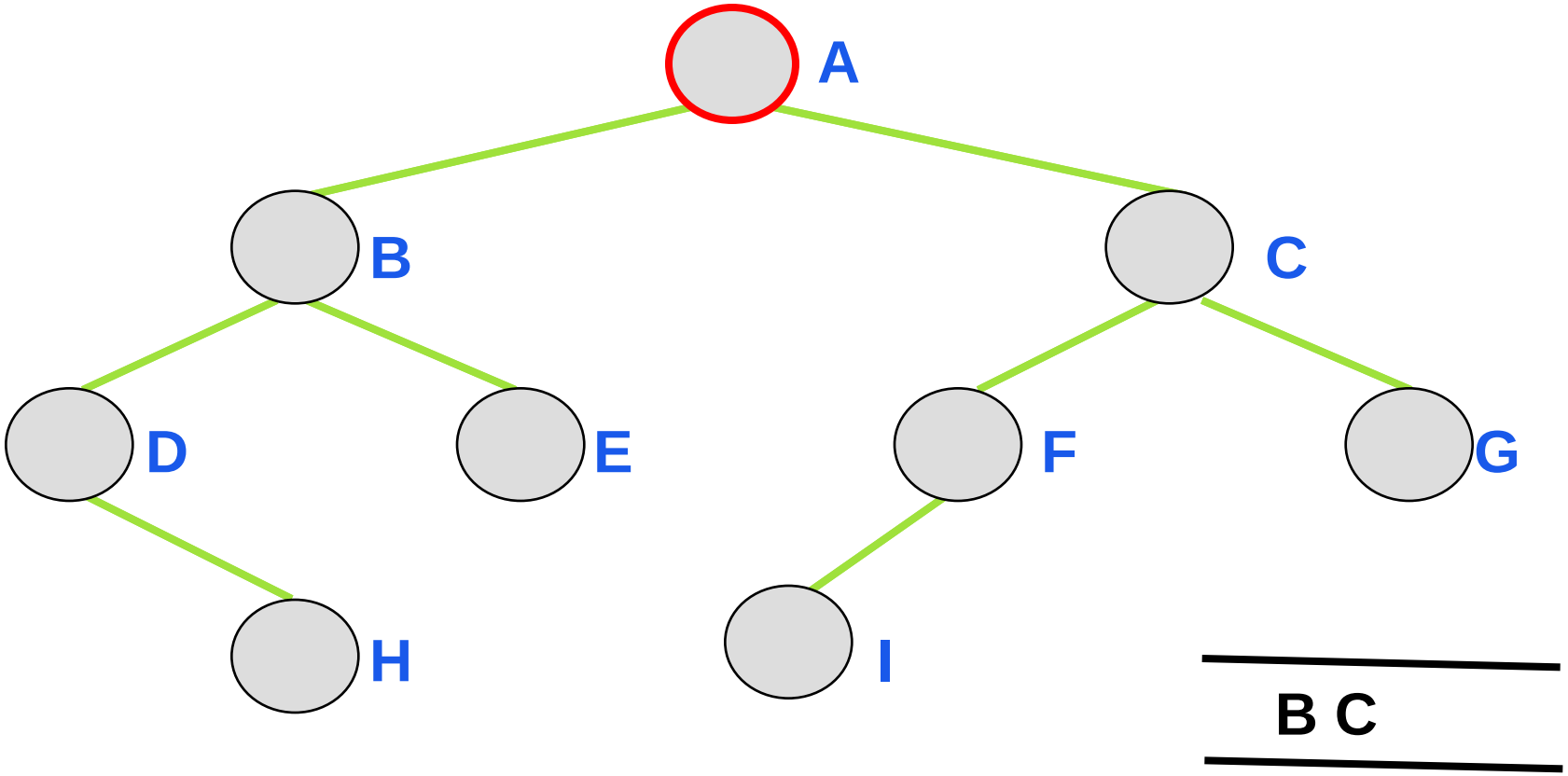
- Start at the root
- Visit the nodes at each level, from left to right



A

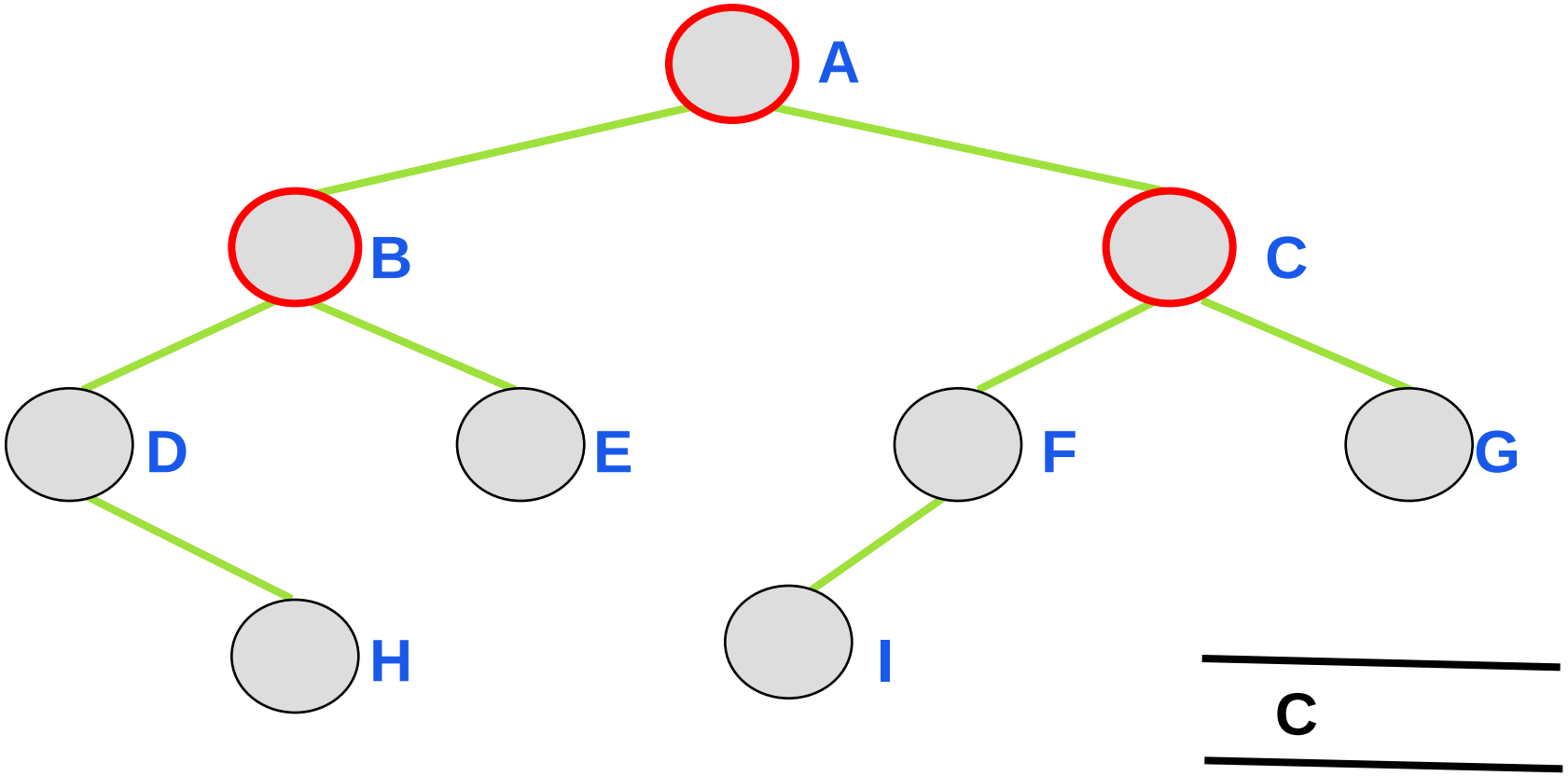
Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



Level Order Traversal

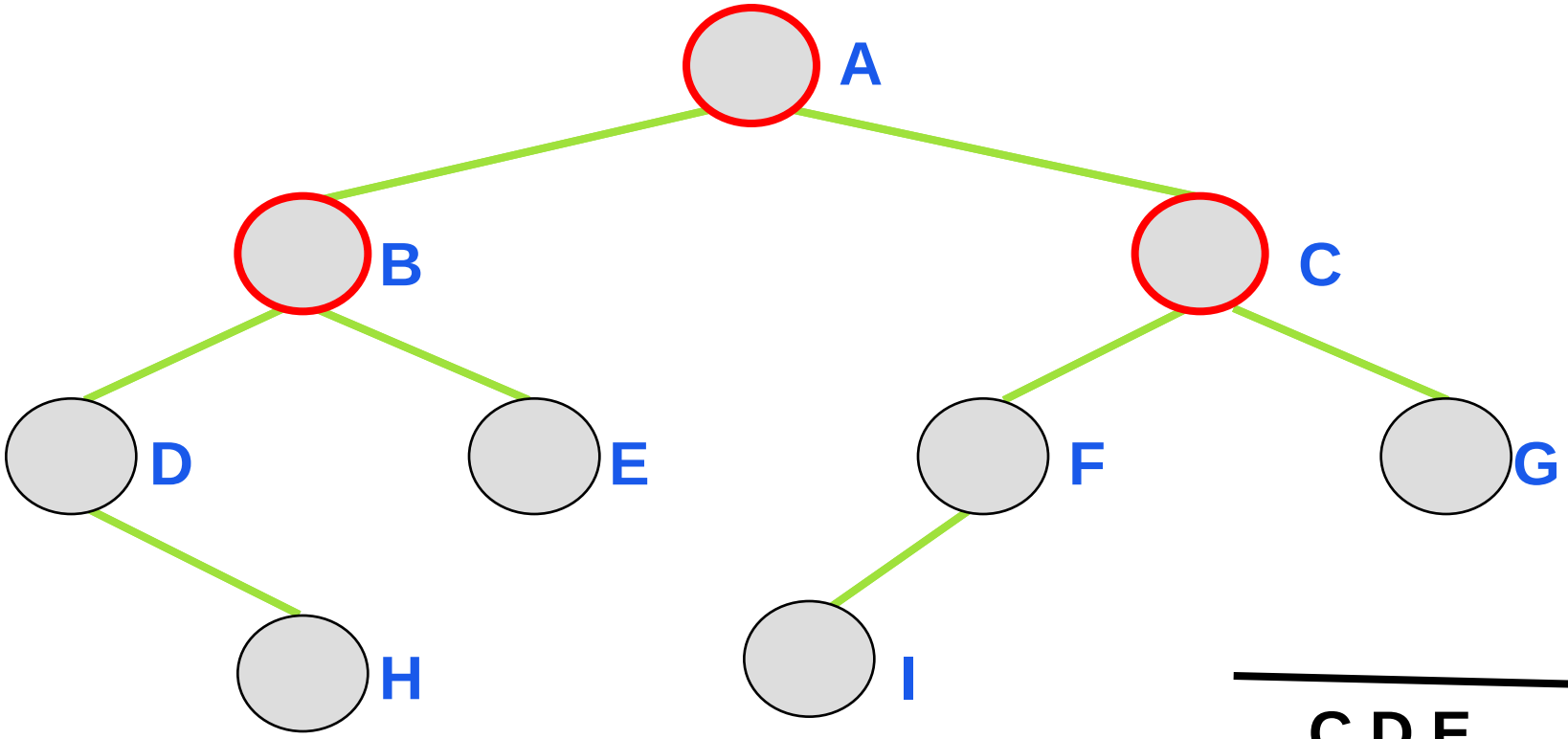
- Start at the root
- Visit the nodes at each level, from left to right



A B

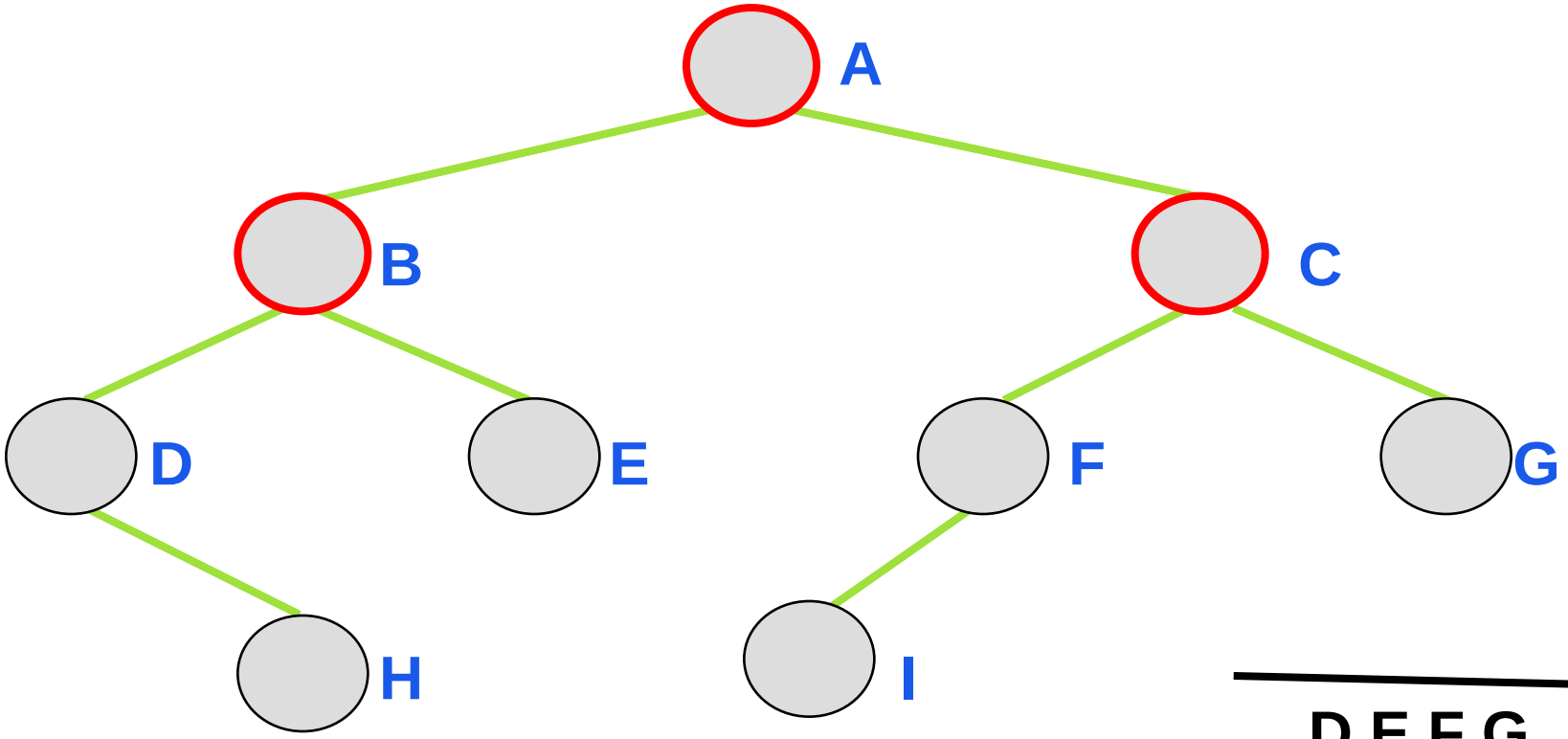
Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right

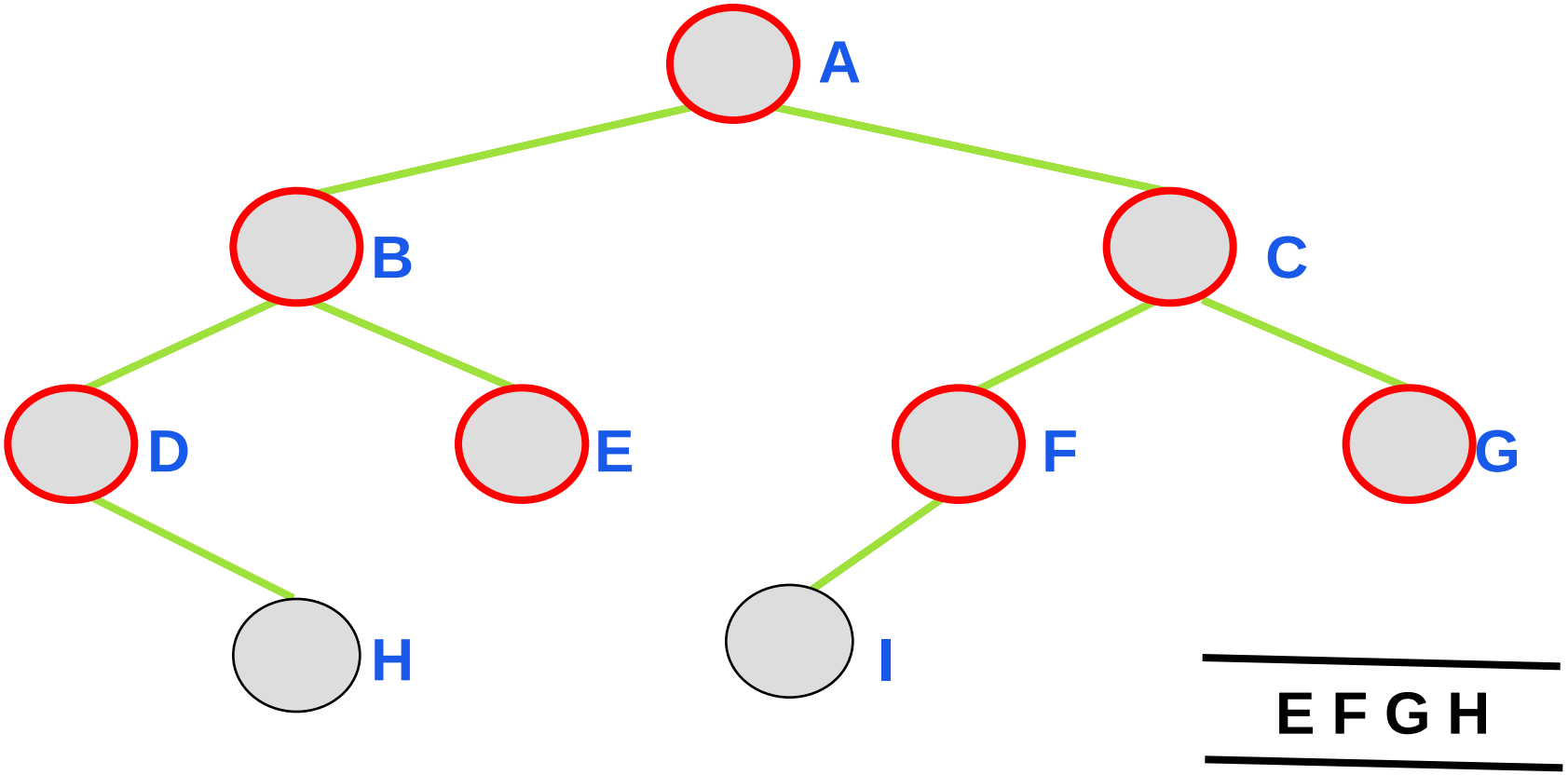


A B C

D E F G

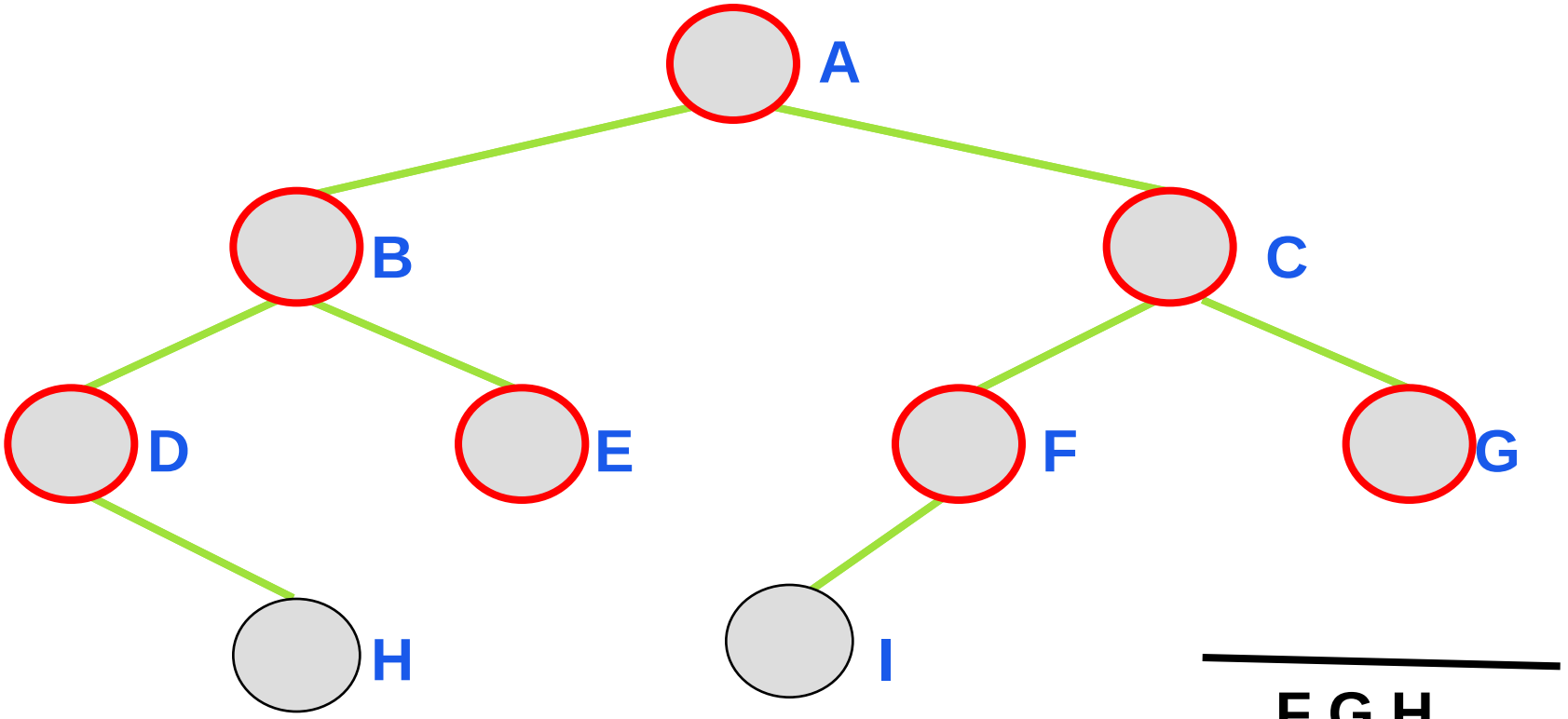
Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



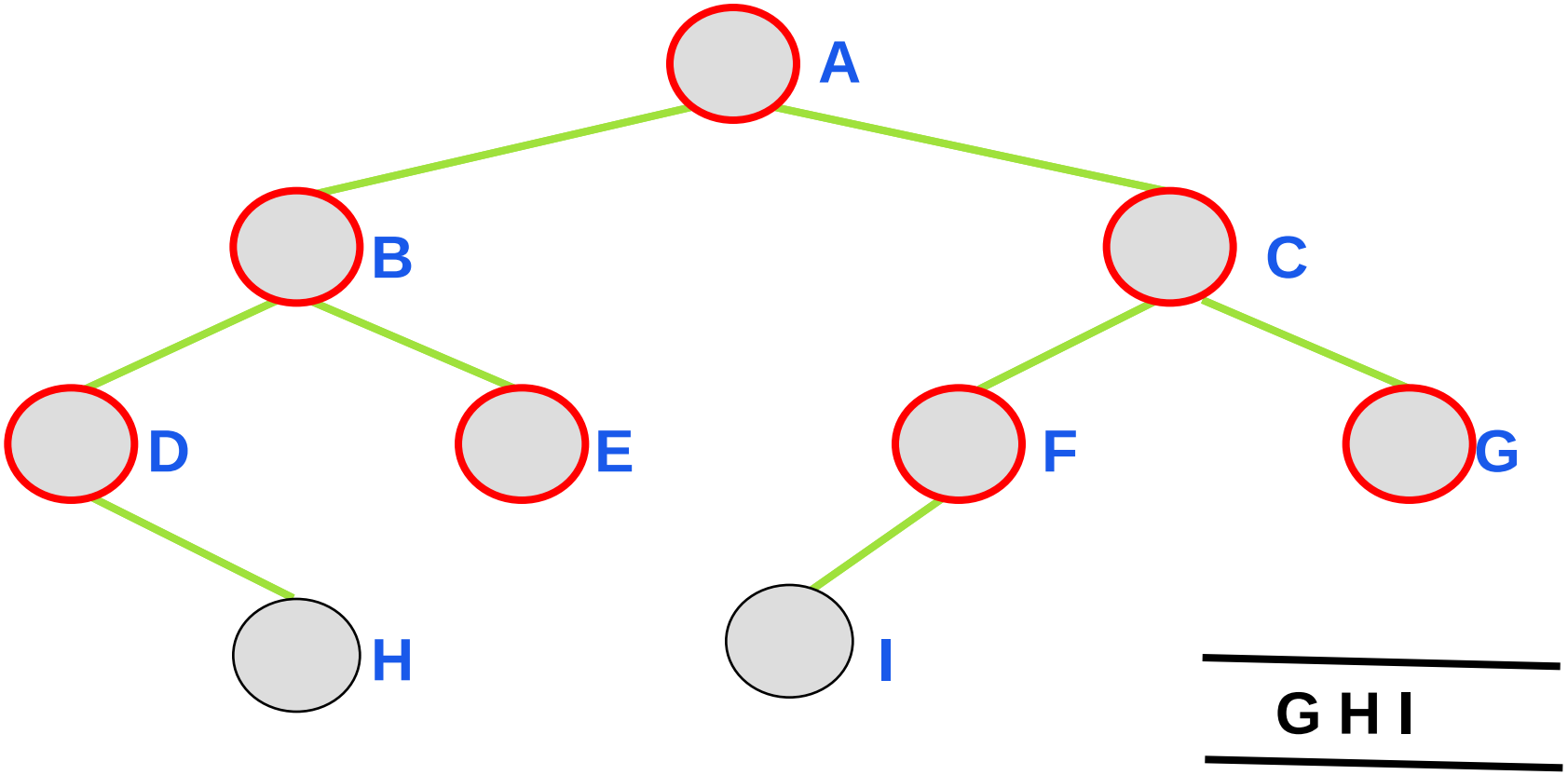
Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



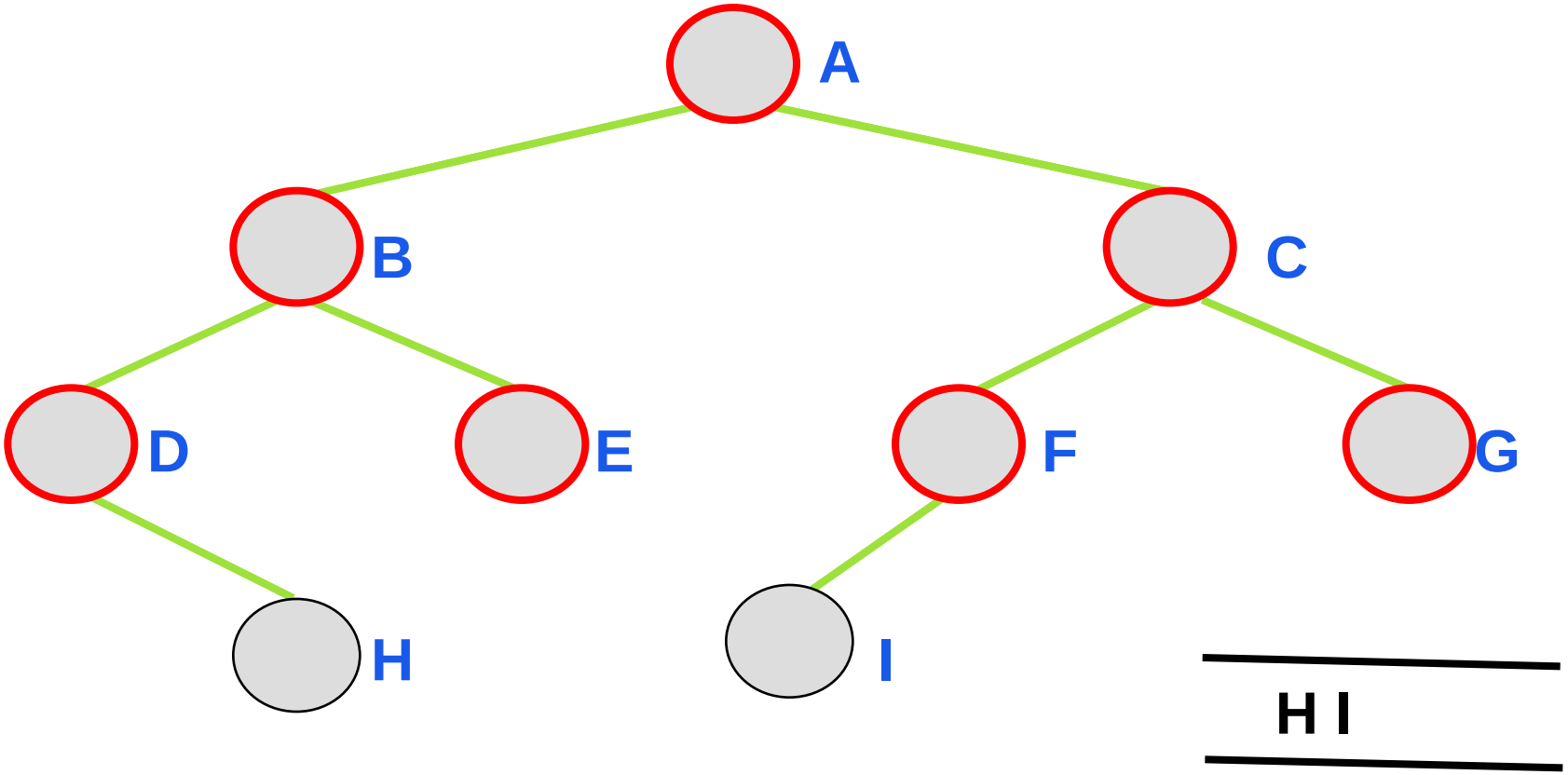
Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



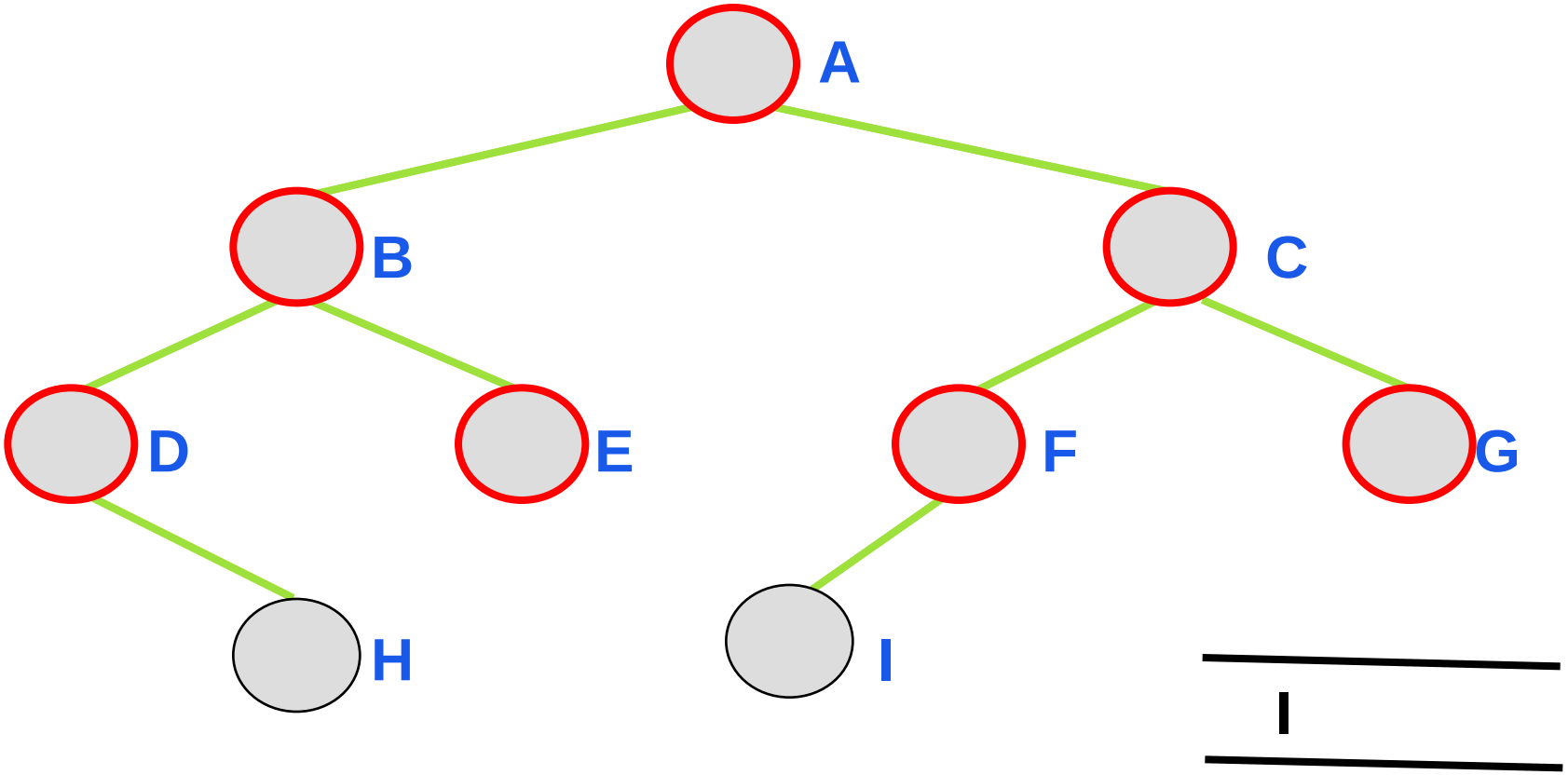
Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



Level Order Traversal

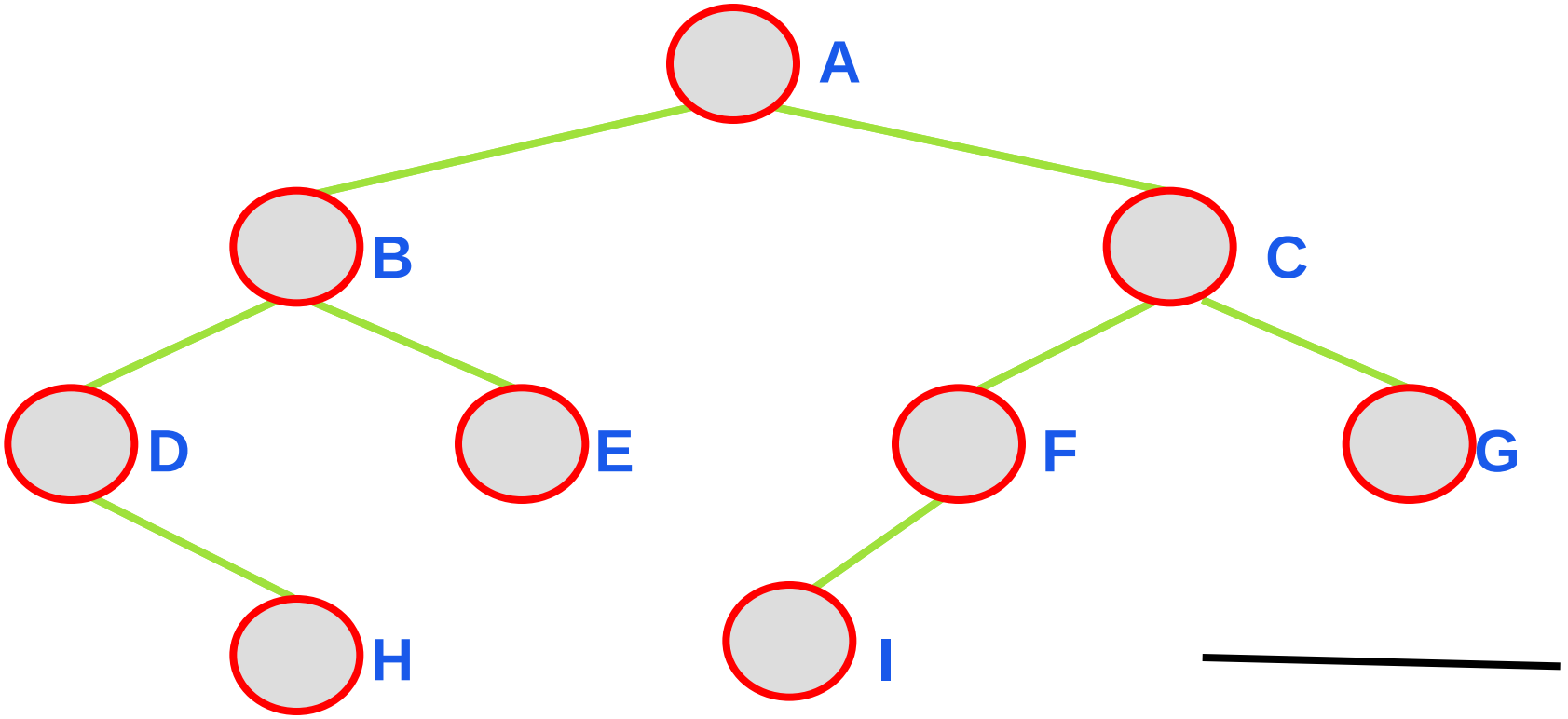
- Start at the root
- Visit the nodes at each level, from left to right



A B C D E F G H I

Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



A B C D E F G H I

Preorder Traversal

- Start at the root
- Visit each node, followed by its children; we will choose to visit left child before right

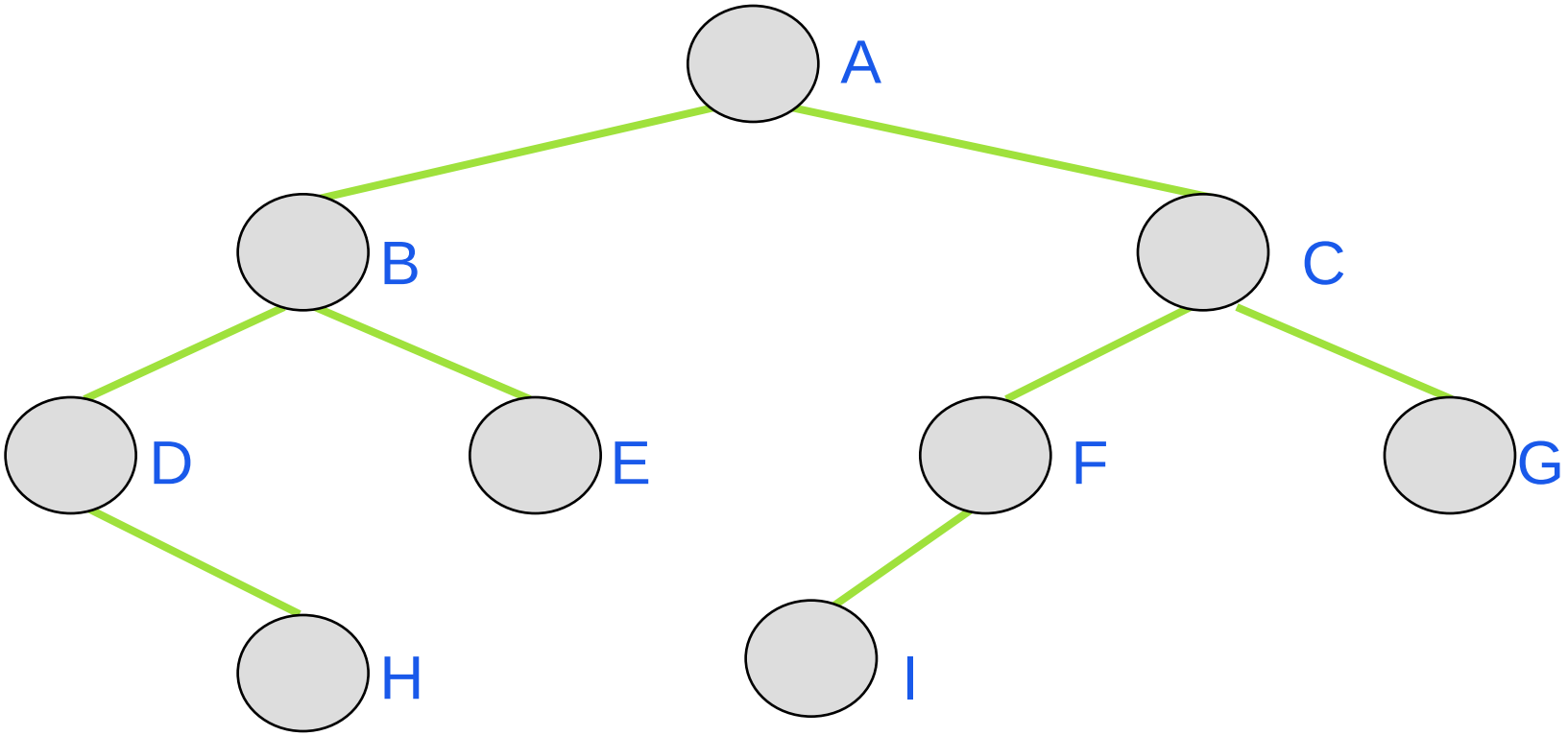
- Recursive algorithm for preorder traversal:
  - If tree is not empty,
    Visit root node of tree
    Perform preorder traversal of its left subtree
    Perform preorder traversal of its right subtree

- What is the base case?
- What is the recursive part?

Preorder Traversal

```
public void preorder (BinaryTreeNode<T> r) {
     if (r != null) {
          visit(r);
          preorder (r.getLeftChild());
          preorder (r.getRightChild());
     }
}
```

Preorder Traversal



Nodes are visited in the order ABDHECFIG

Inorder Traversal

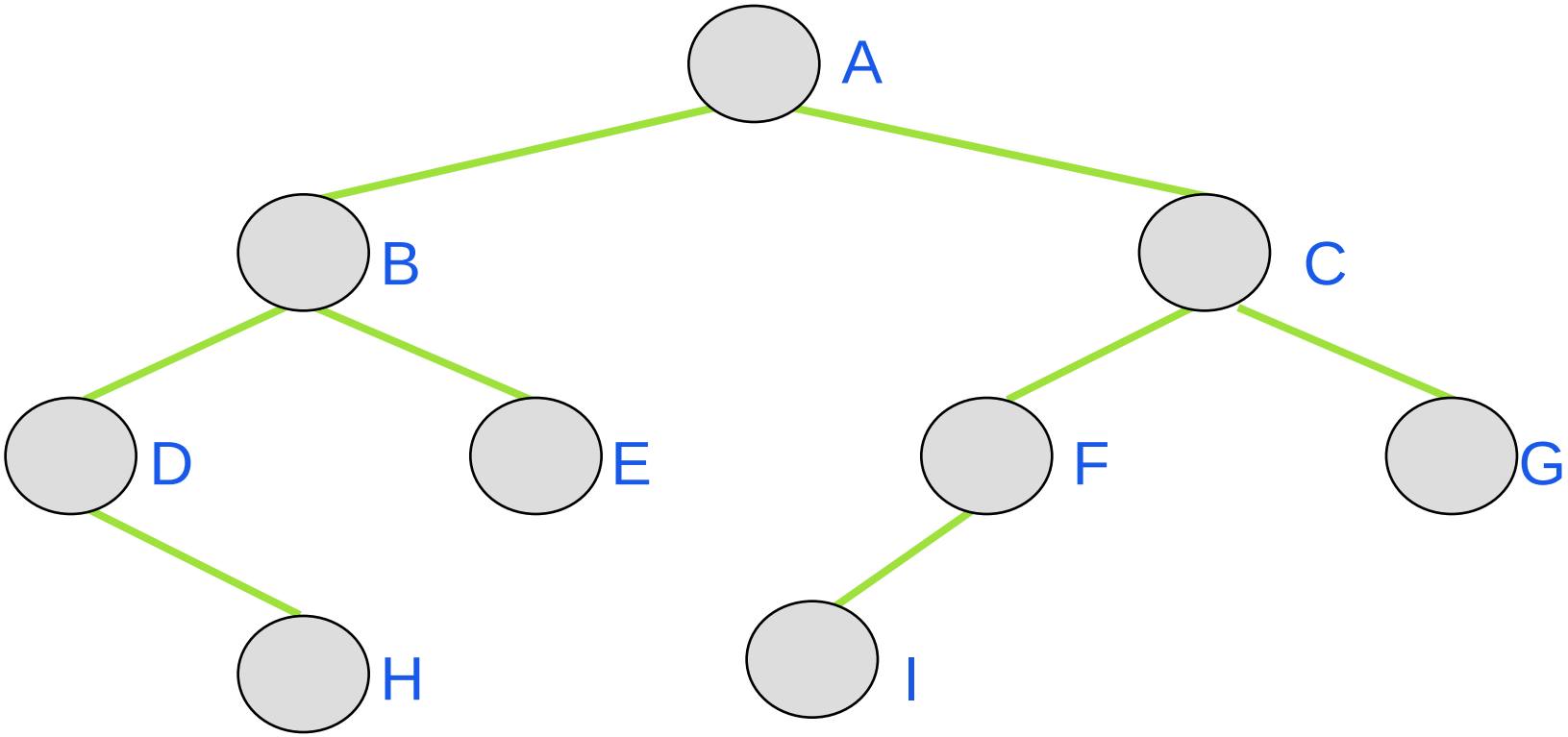
- Start at the root
- Visit the left child of each node, then the node, then any remaining nodes

- Recursive algorithm for inorder traversal
  - If tree is not empty,
    Perform inorder traversal of left subtree of root
    Visit root node of tree
    Perform inorder traversal of its right subtree

Inorder Traversal

```
public void inorder (BinaryTreeNode<T> r) {
     if (r != null) {
          inorder (r.getLeftChild());
          visit(r);
          inorder (r.getRightChild());
     }
}
```

Inorder Traversal



Nodes are visited in the order **DHBEAIFCG**

Postorder Traversal

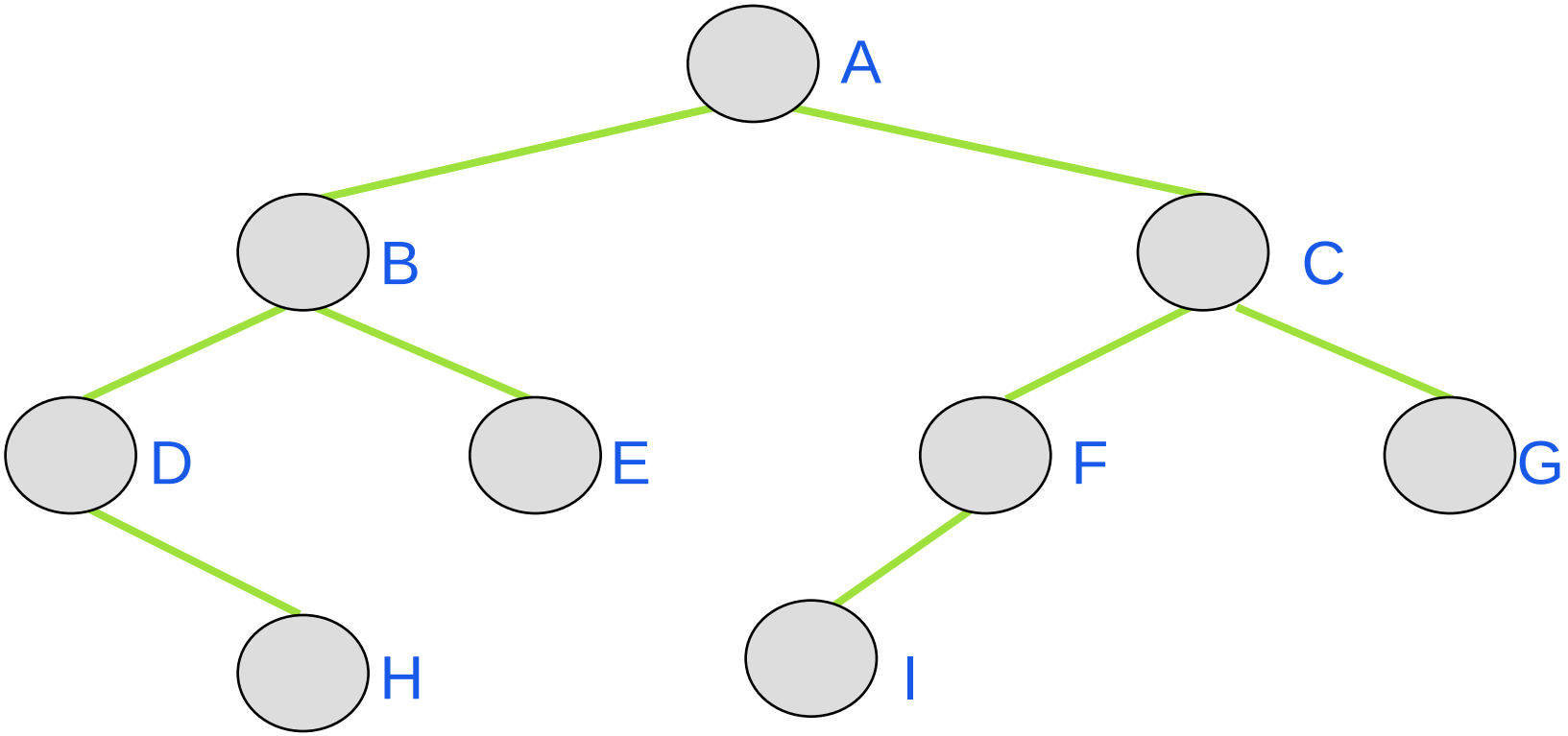
- Start at the root
- Visit the children of each node, then the node

- Recursive algorithm for postorder traversal
  - If tree is not empty,
    Perform postorder traversal of left subtree of root
    Perform postorder traversal of right subtree of root
    Visit root node of tree

Postorder Traversal

```
public void postorder (BinaryTreeNode<T> r) {
     if (r != null) {
          postorder (r.getLeftChild());
          postorder (r.getRightChild());
          visit(r);
     }
}
```

Postorder Traversal



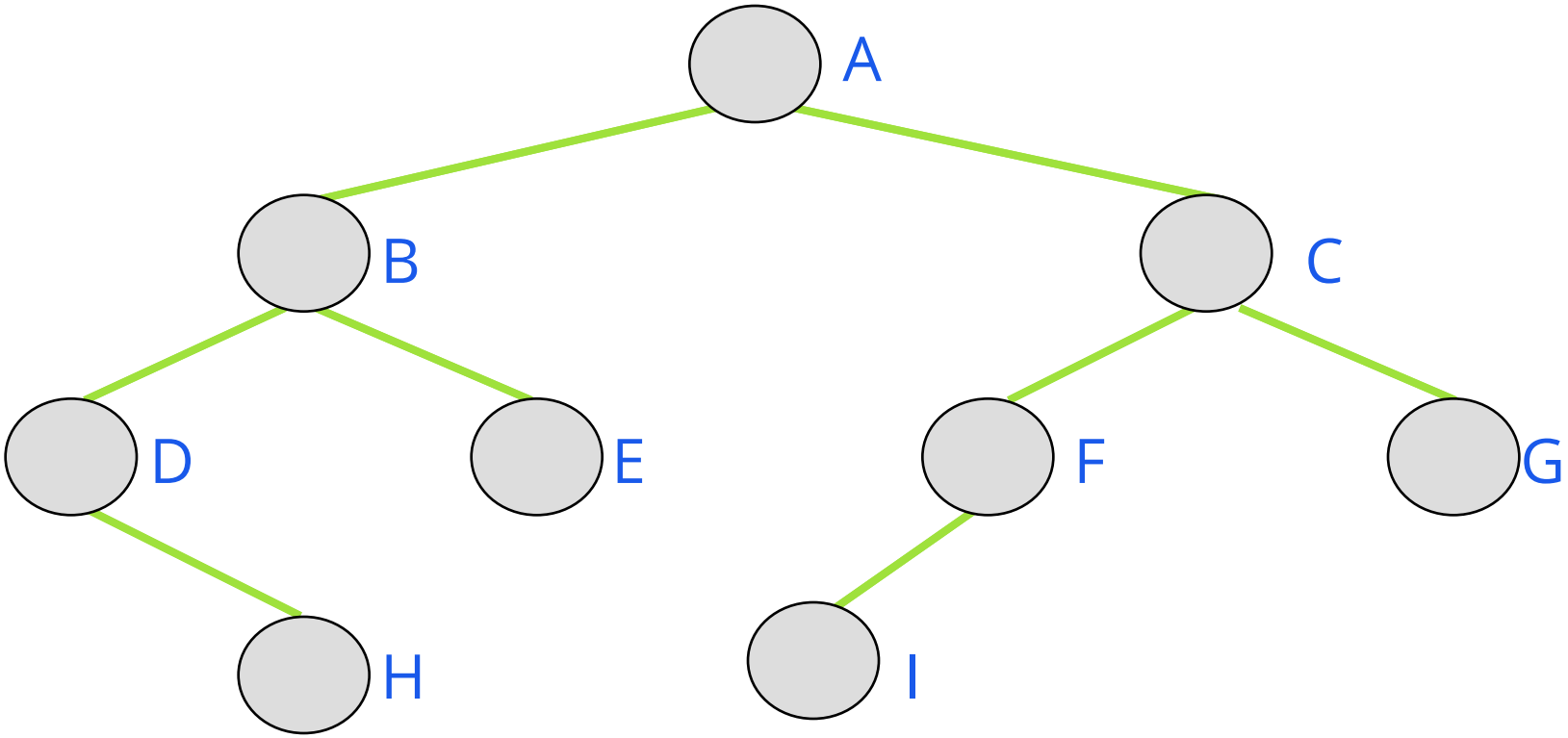
Nodes are visited in the order **HDEBIFGCA**

Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right

- Is there a recursive algorithm for a level order traversal?

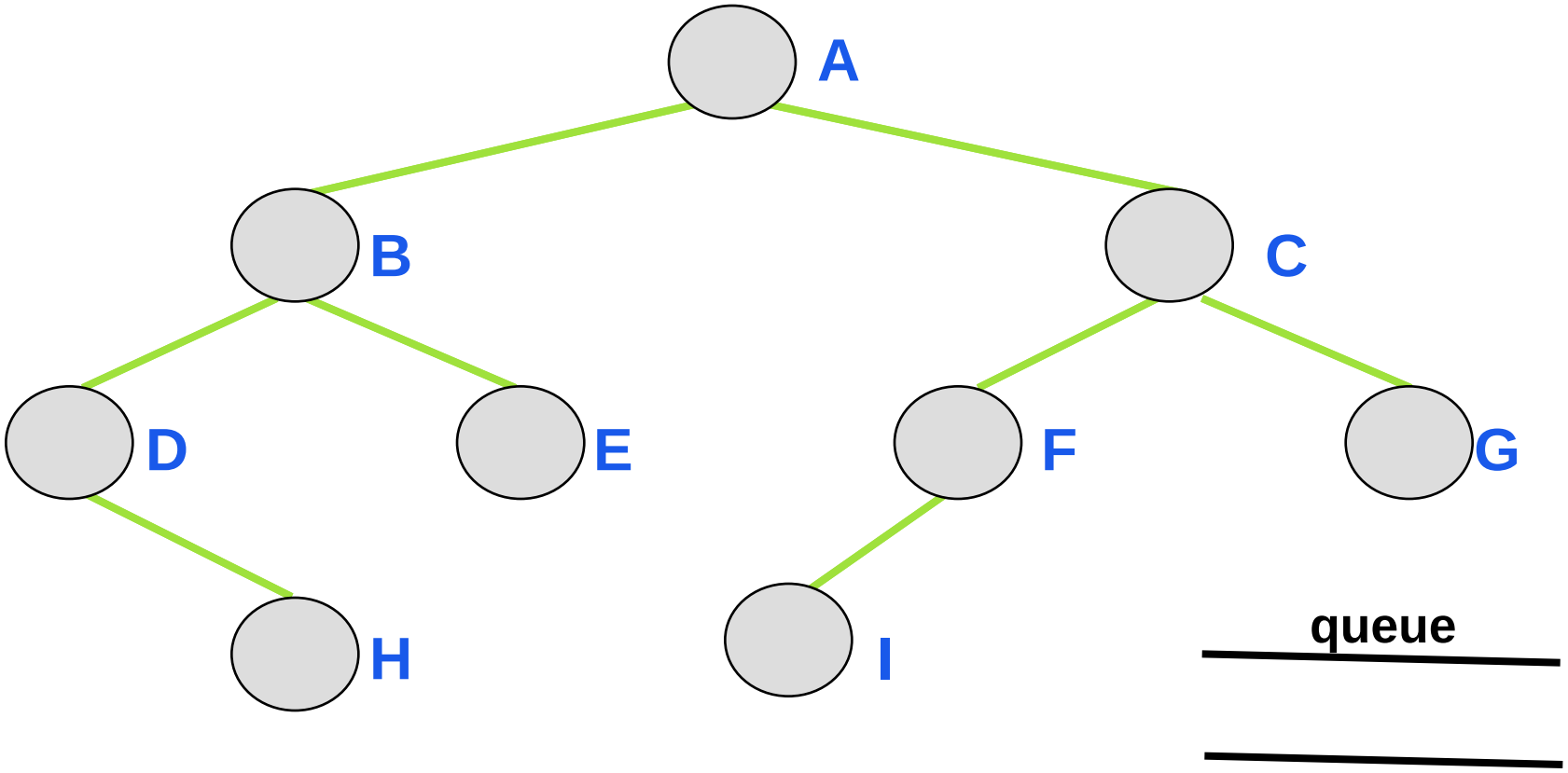
Level Order Traversal



Nodes will be visited in the order ABCDEFGHI

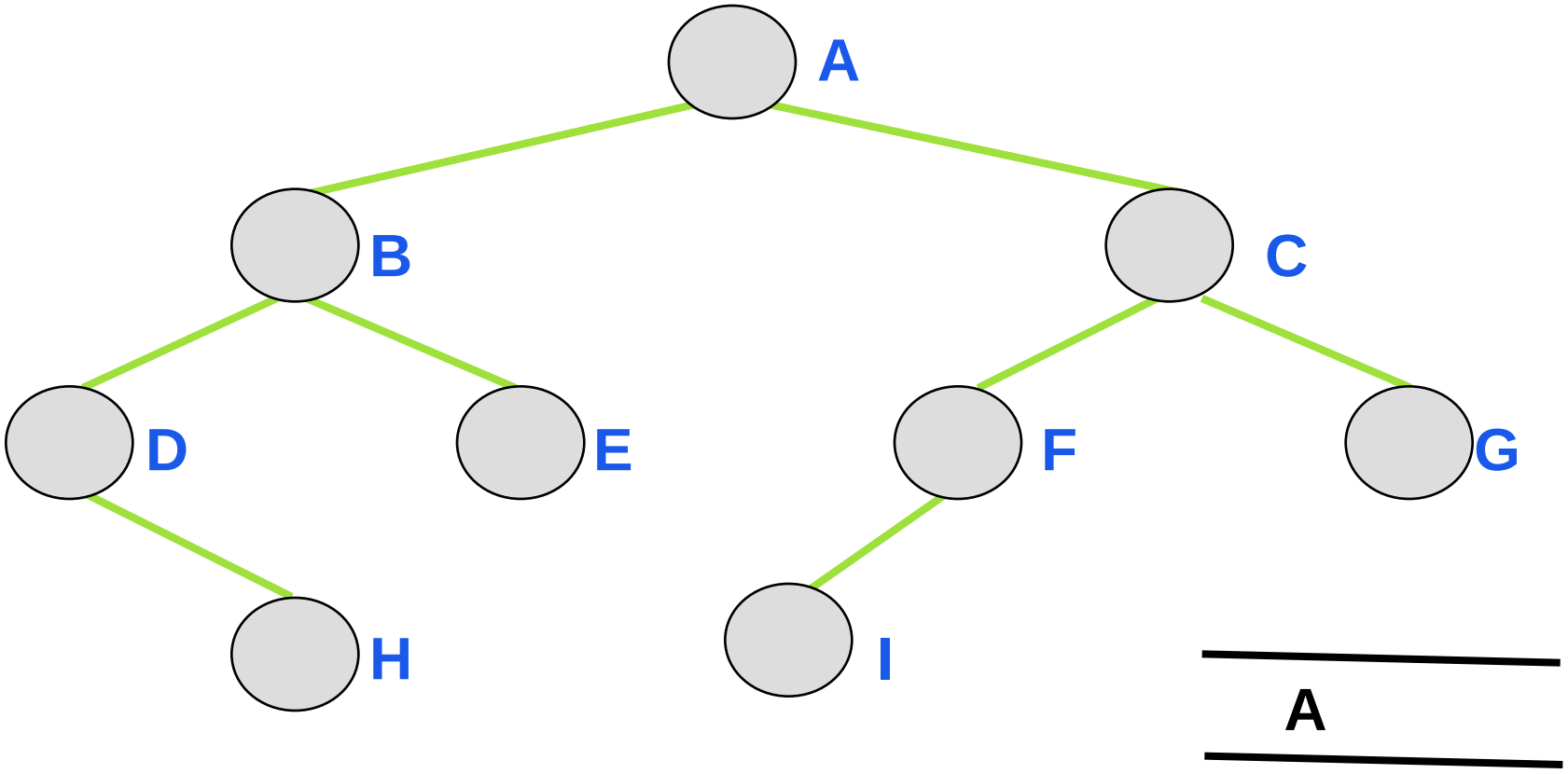
Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



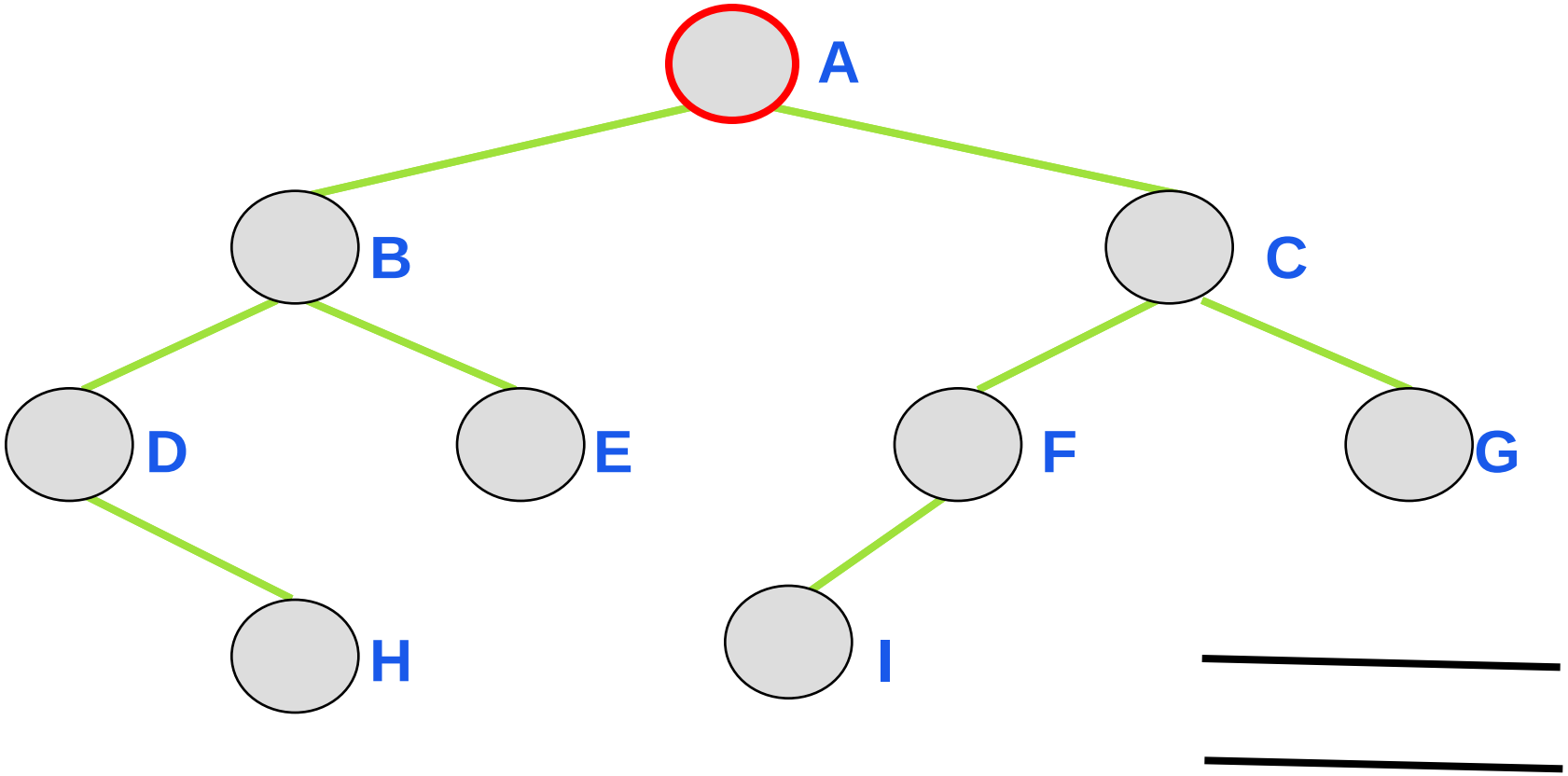
Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



Level Order Traversal

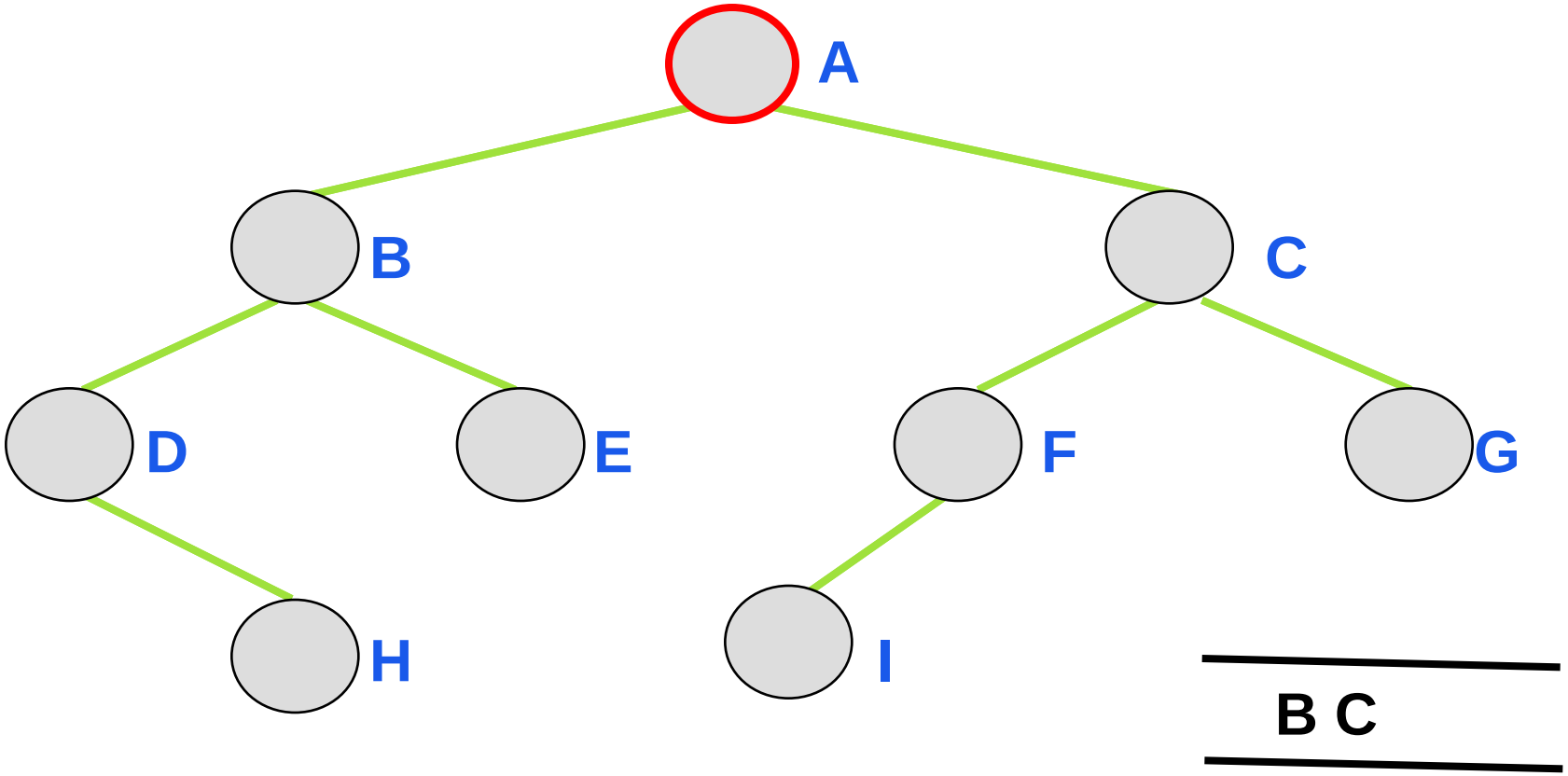
- Start at the root
- Visit the nodes at each level, from left to right



A

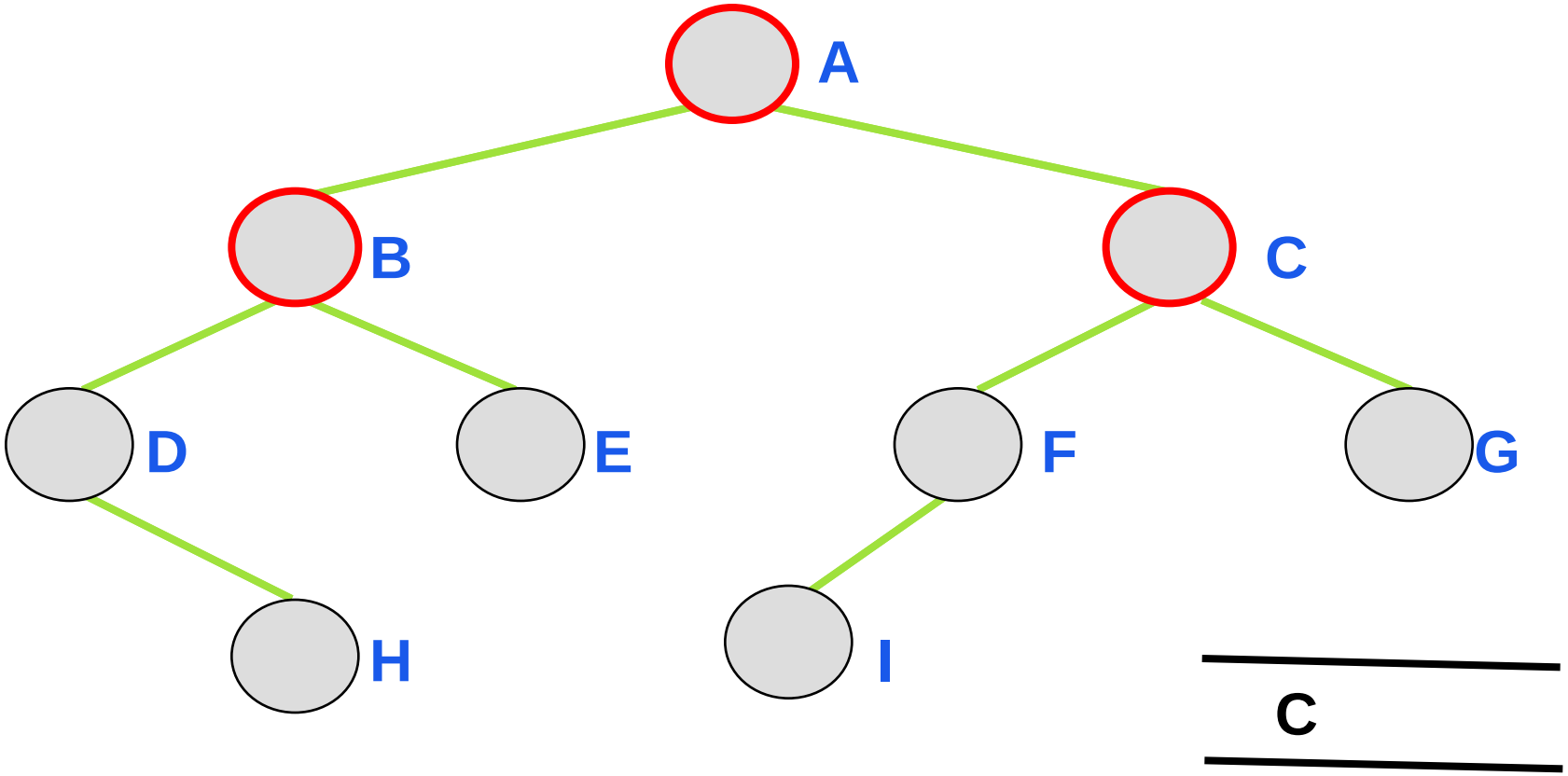
Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



Level Order Traversal

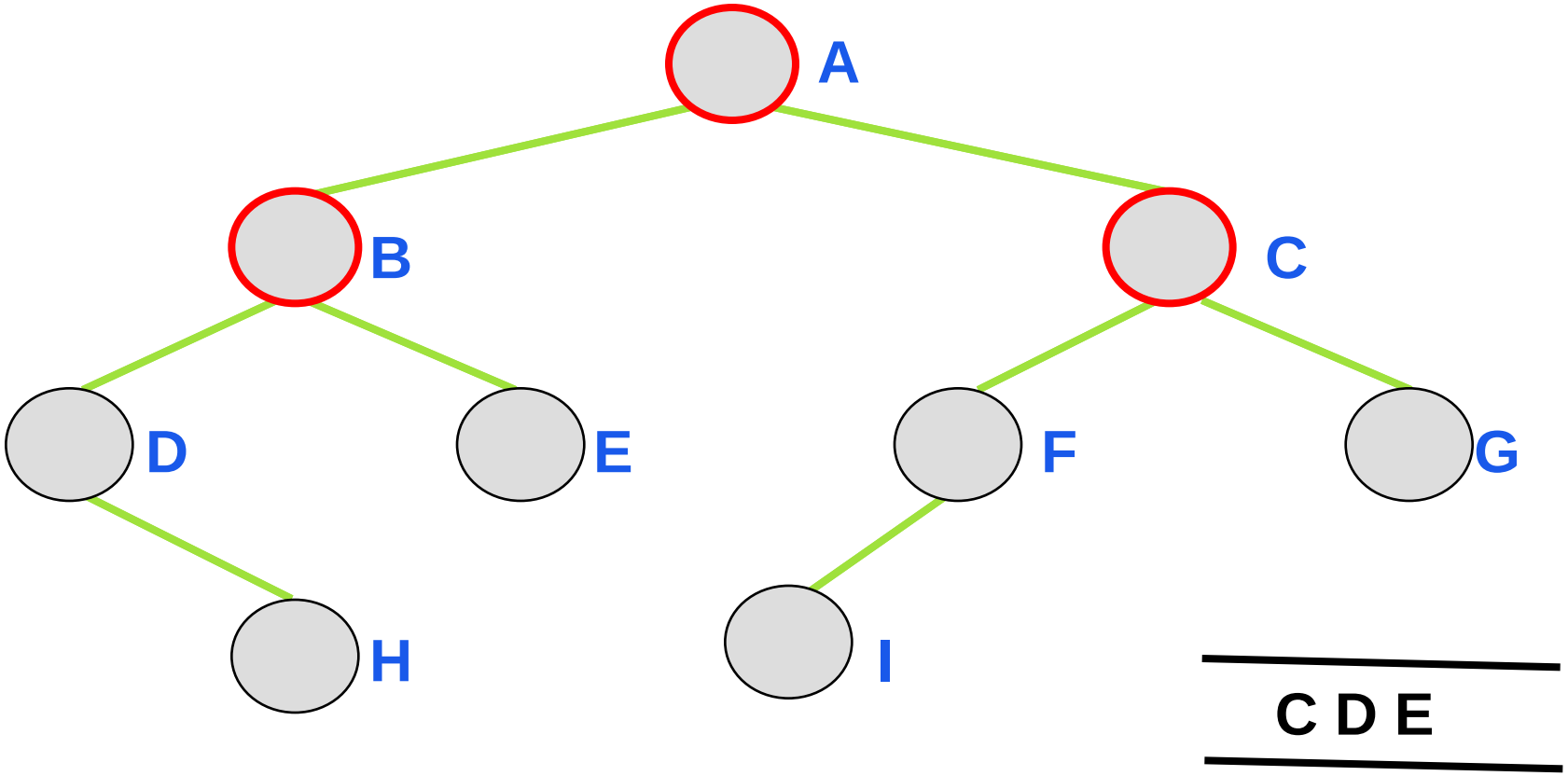
- Start at the root
- Visit the nodes at each level, from left to right



A B

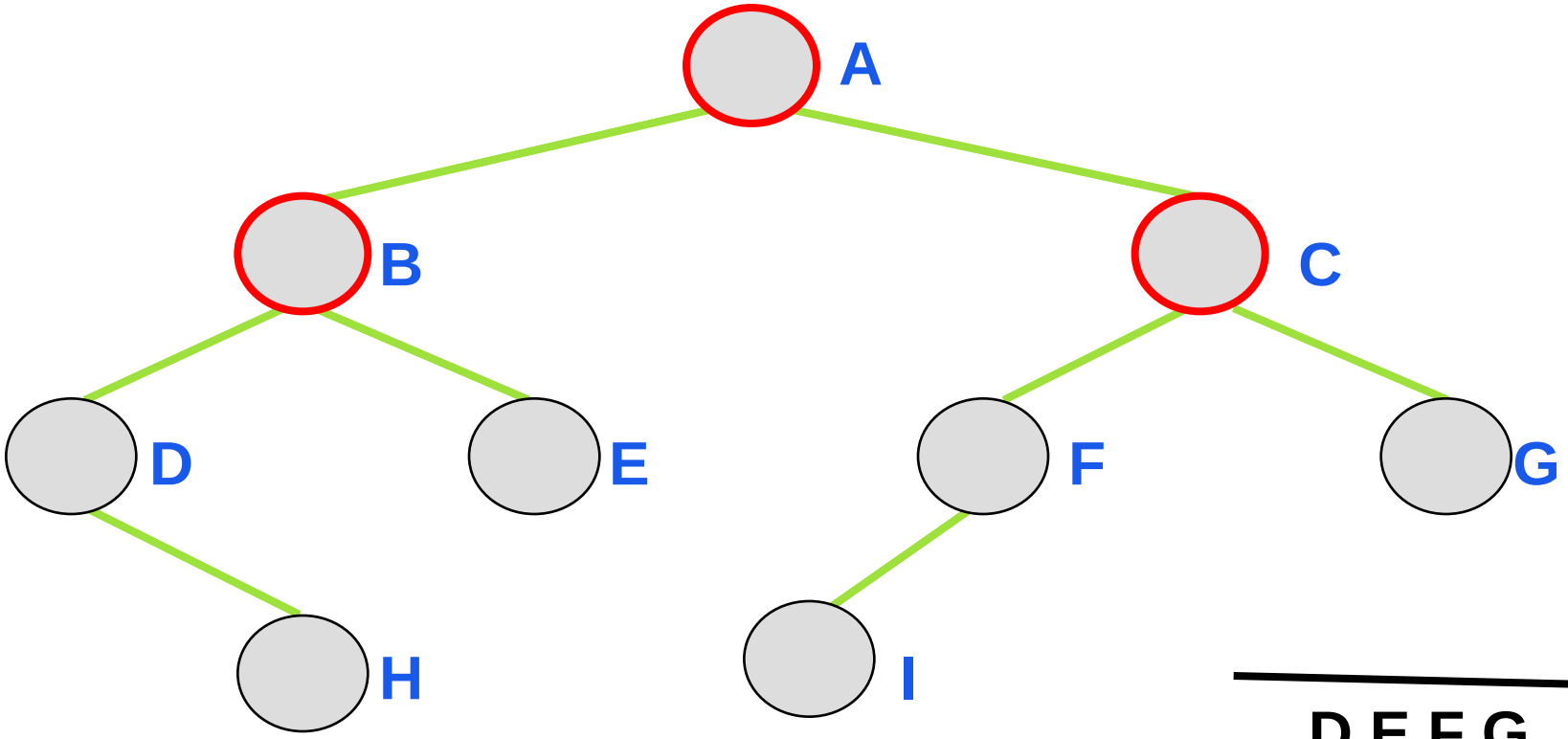
Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



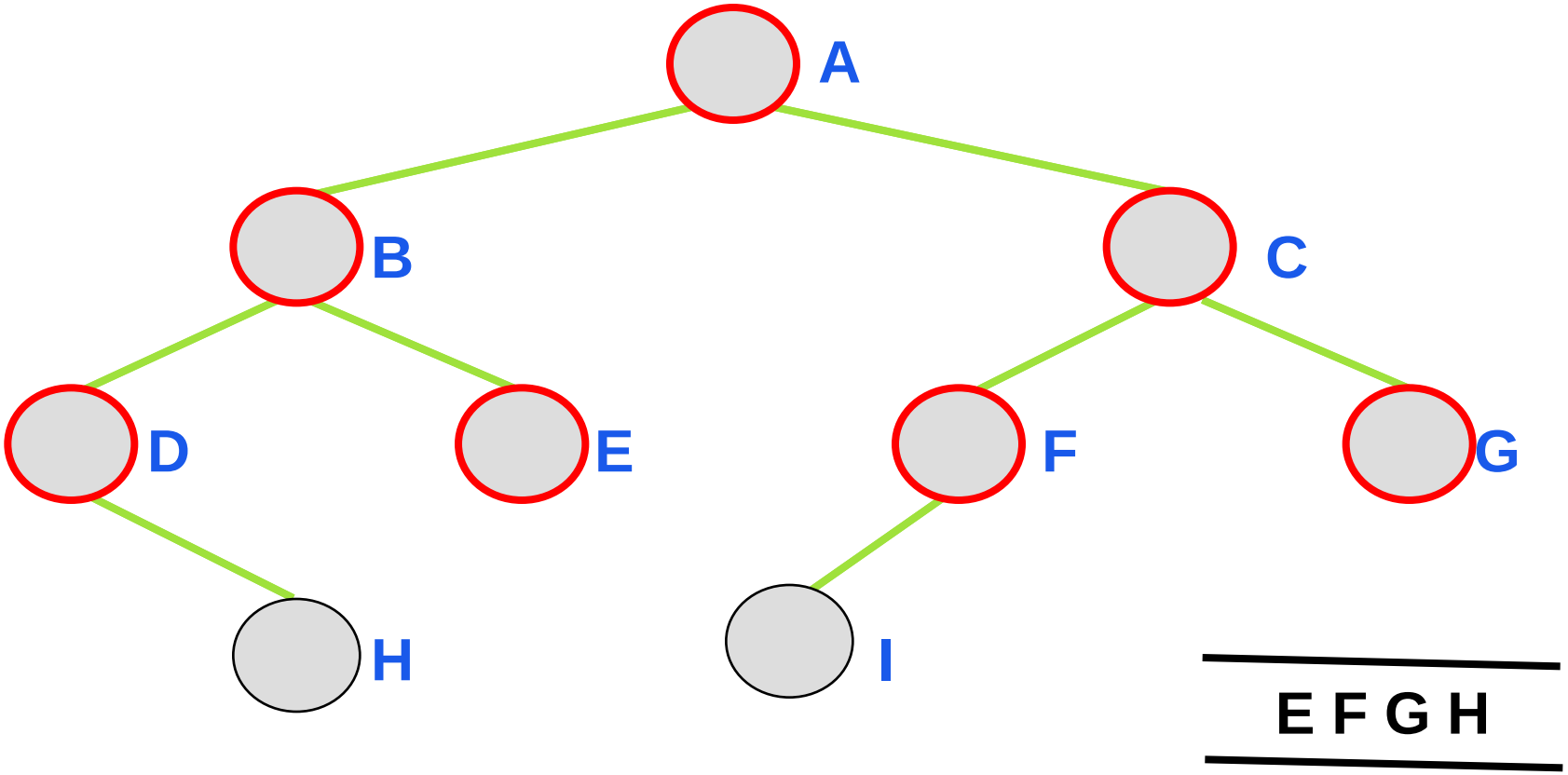
Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



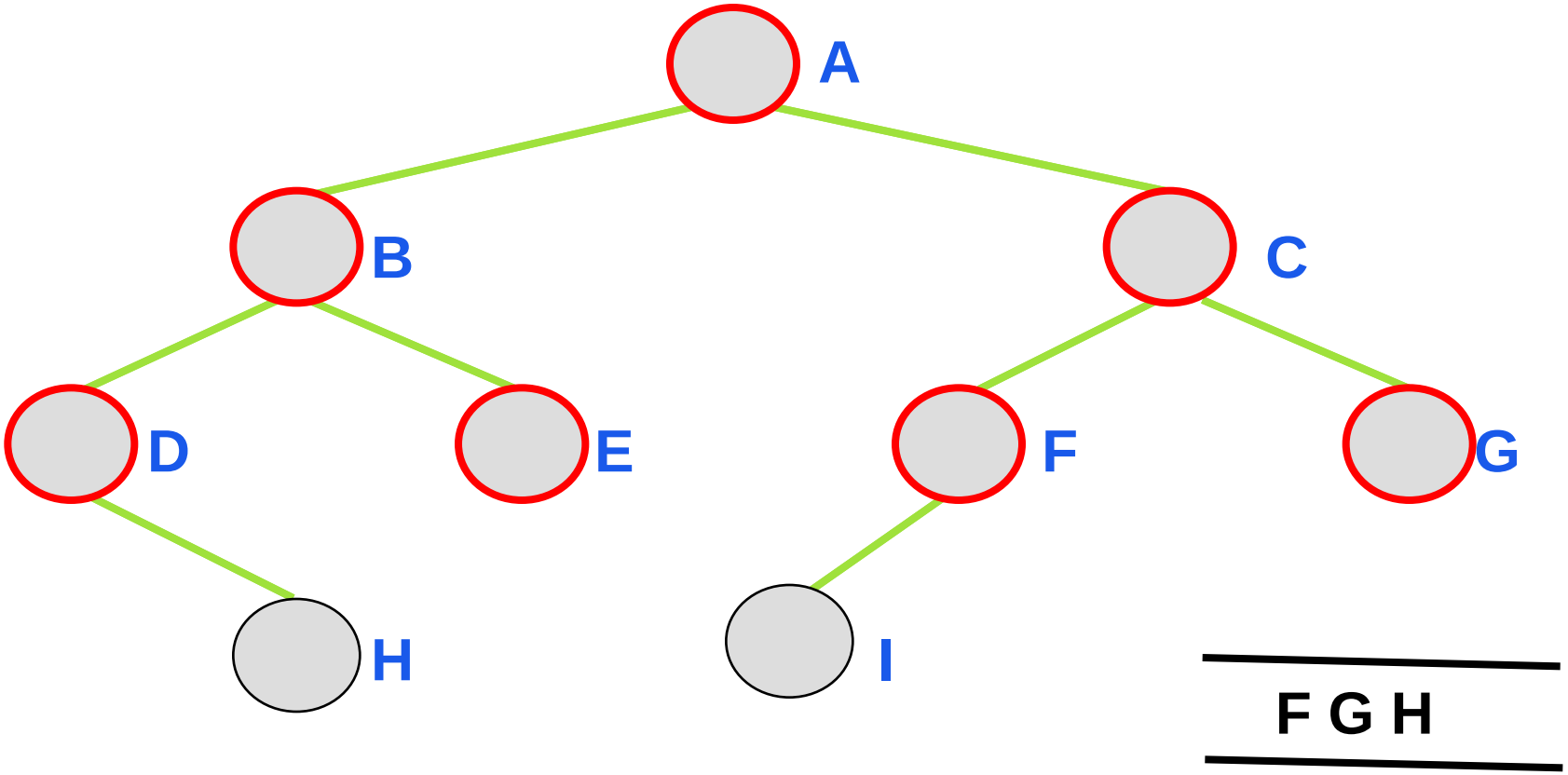
Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



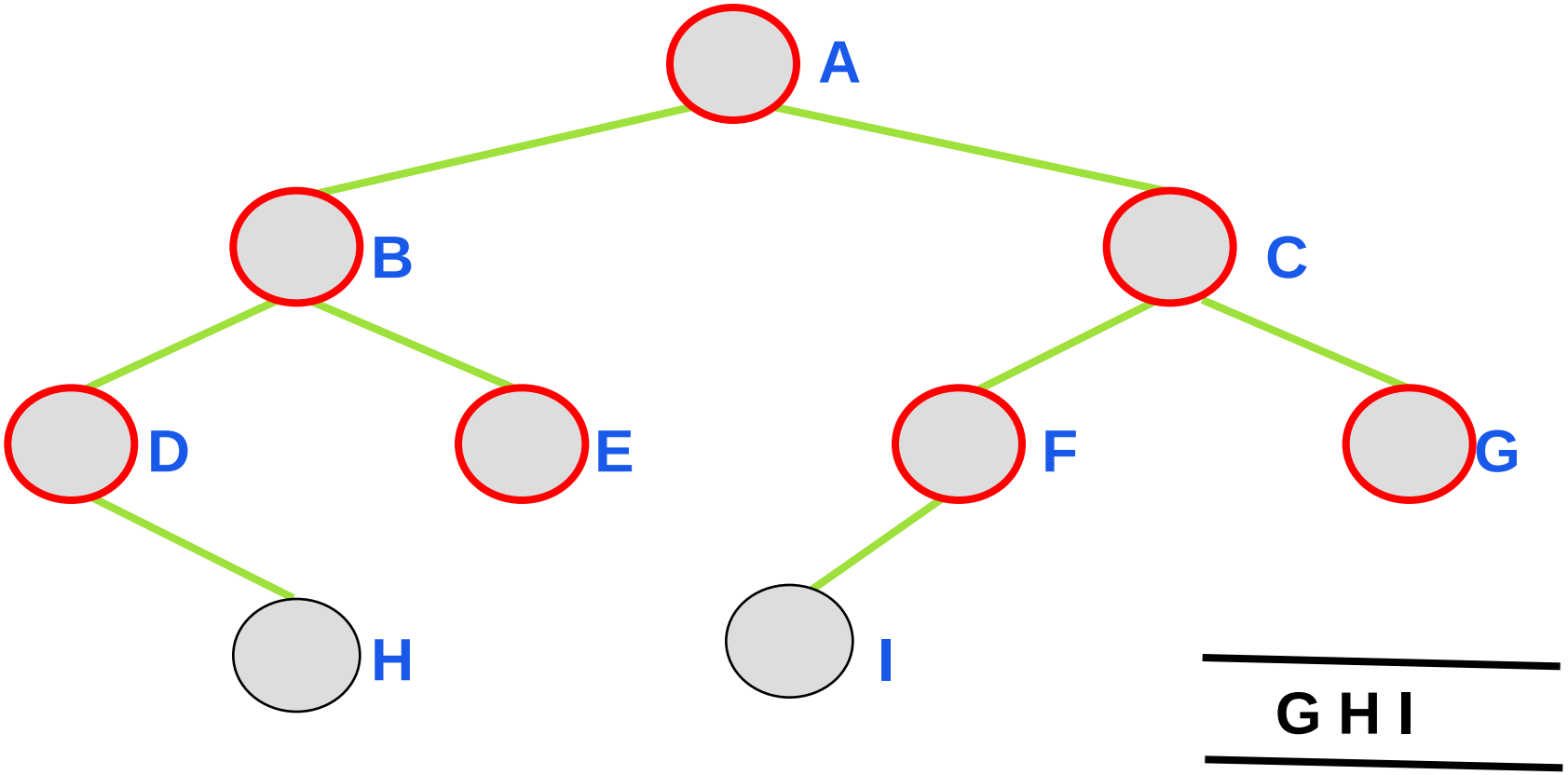
Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



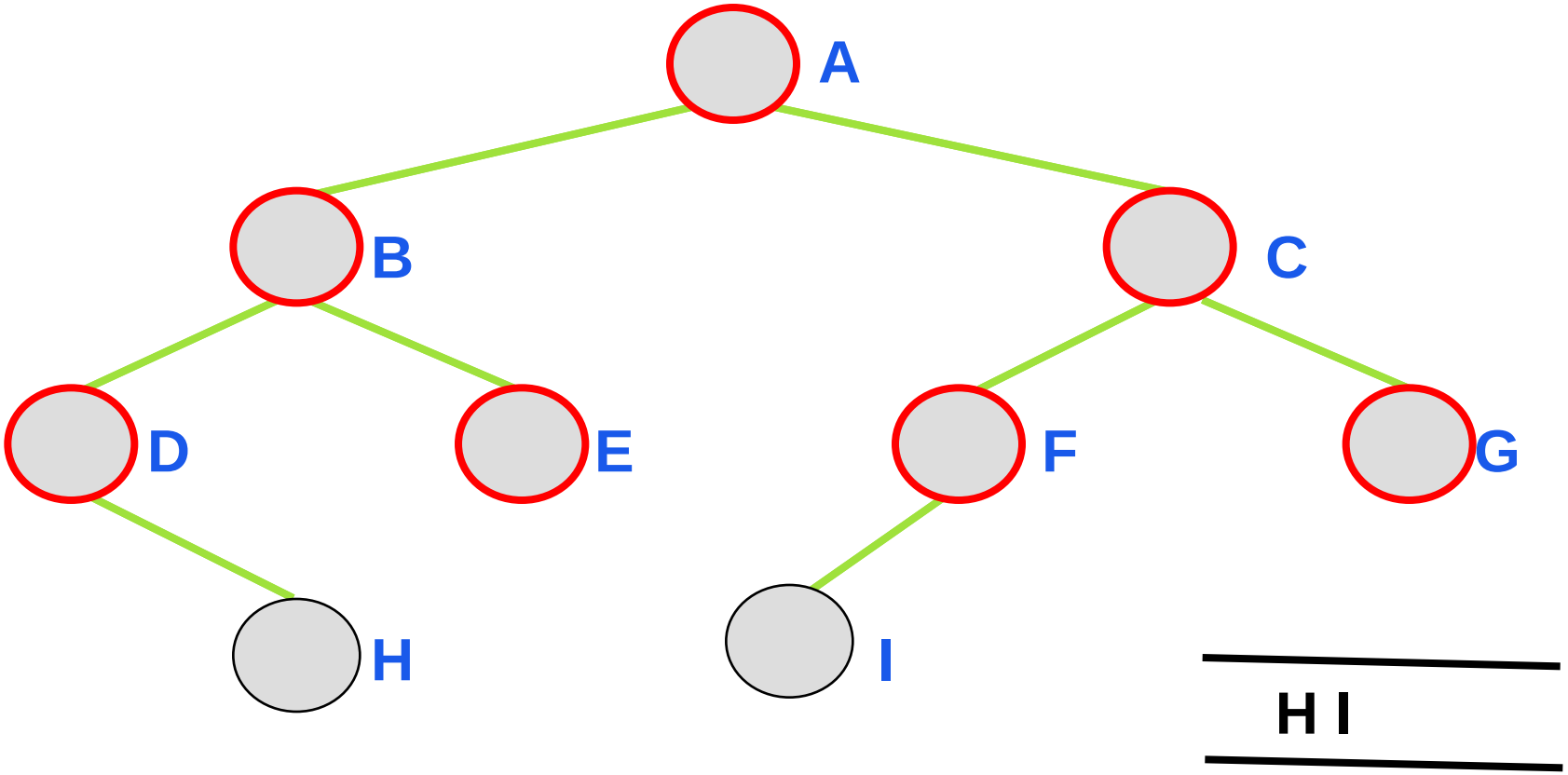
Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



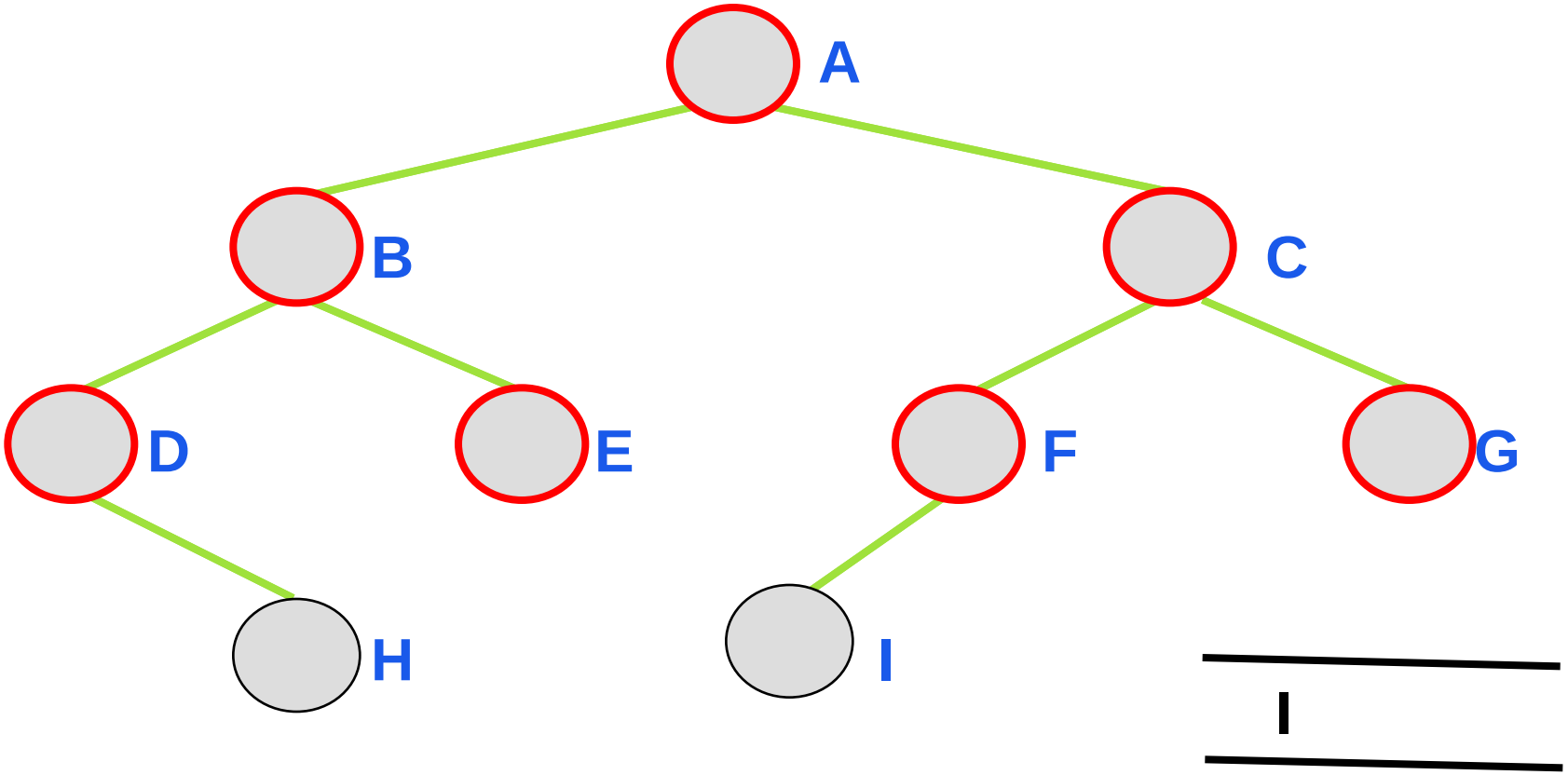
Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



Level Order Traversal

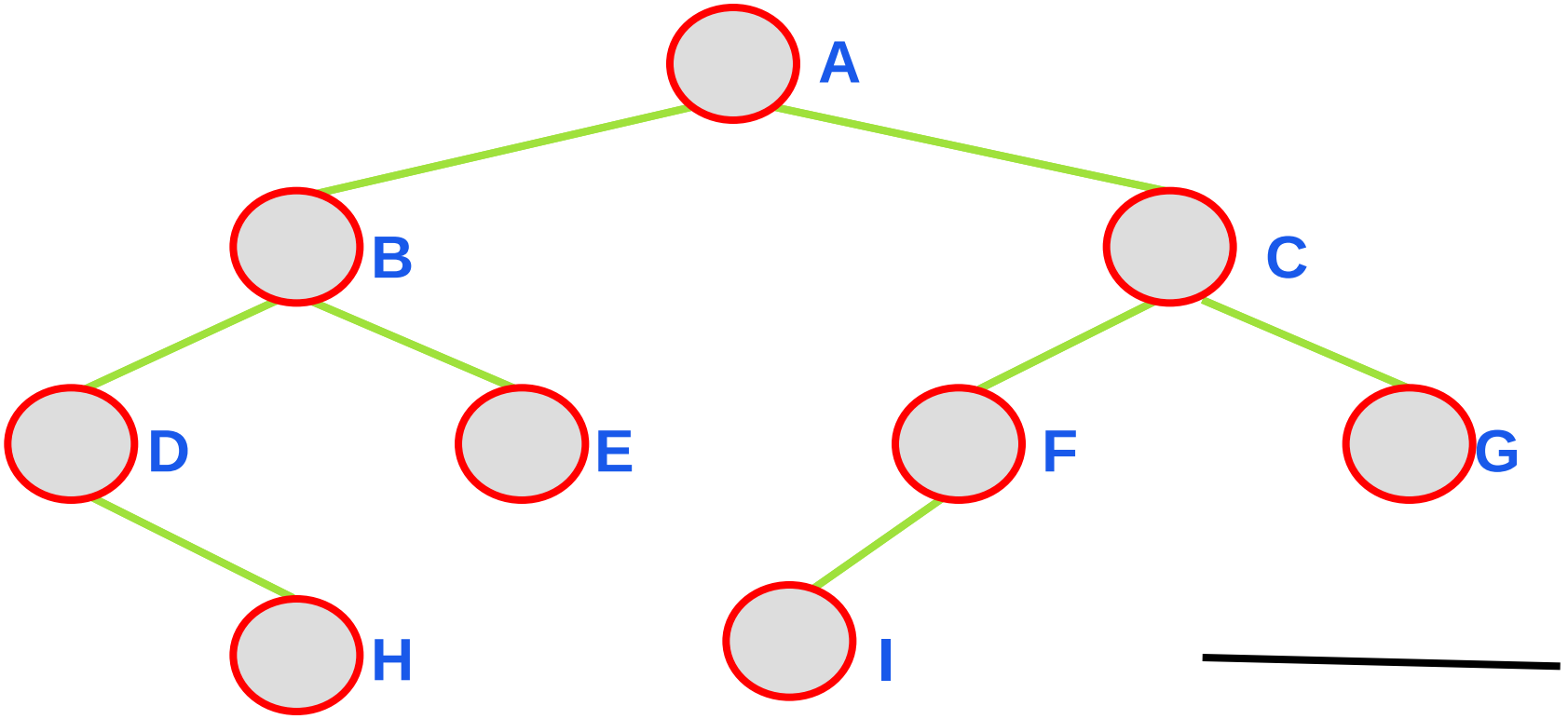
- Start at the root
- Visit the nodes at each level, from left to right



A B C D E F G H

Level Order Traversal

- Start at the root
- Visit the nodes at each level, from left to right



A B C D E F G H I

Level Order Traversal

```
public void levelOrder (BinaryTreeNode<T> root) {
     if (root == null) return;
     LinkedQueue<T> Q = new LinkedQueue<T>();
     Q.enqueue(root);
     while (!Q.isEmpty()) {
          BinaryTreeNode<T> v = Q.dequeue();
          visit(v);
          if (v.leftChild() != null)
             Q.enqueue(v.leftChild());
          if (v.rightChild() != null)
             Q.enqueue(v.rightChild());
     }
}
```

Questions?

Slide Credit

- Java Software Structures: Designing and Using Data Structures, Lewis and Chase, Addison Wesley, 4th Edition.
- Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014
- Data Structures: Abstraction and Design Using Java. Elliot B. Koffman and Paul A. T. Wolfgang