Binary Trees

Reading:
Lewis & Chase 12.1, 12.3
Eck 9.4.1

Objectives

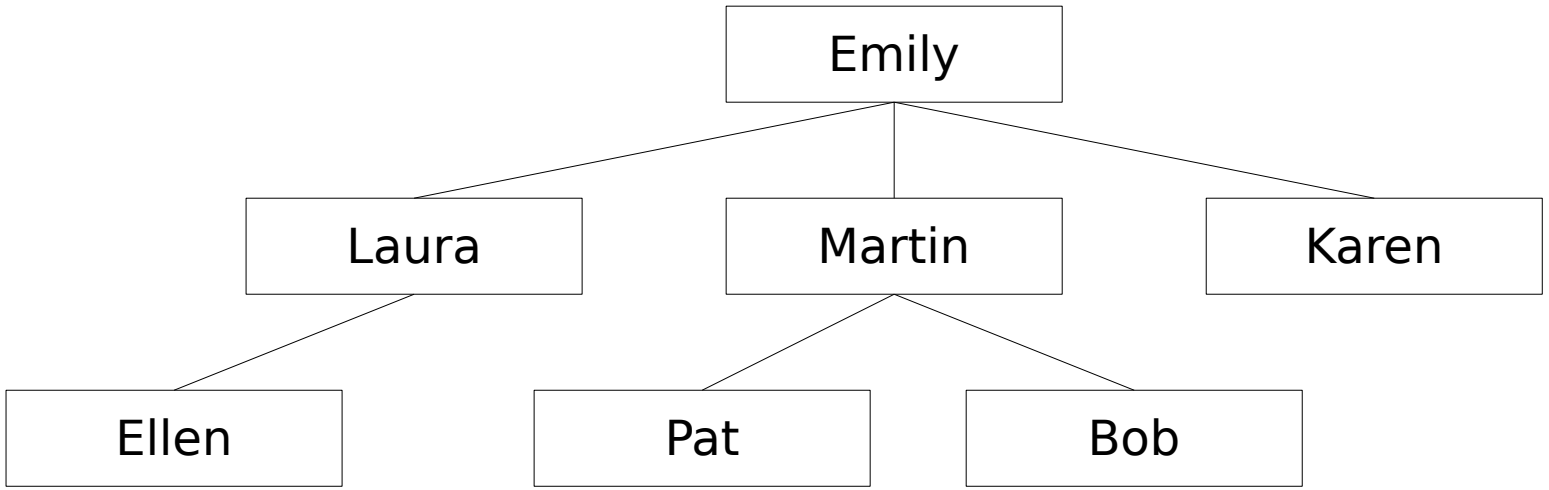
- Tree basics
- Learn the terminology used when talking about trees
- Discuss methods for traversing trees
- Discuss a possible implementation of trees with nodes
- Examine a binary tree example

Tree Basics

- So far, all the data structures that we have encountered were linear. Objects in an array, list, stack, or queue are placed one after the other in a line.
- Sometimes it is useful to organize data into groups and subgroups.
- This type of organization of data is hierarchical, or non-linear, since the data appears at various levels.

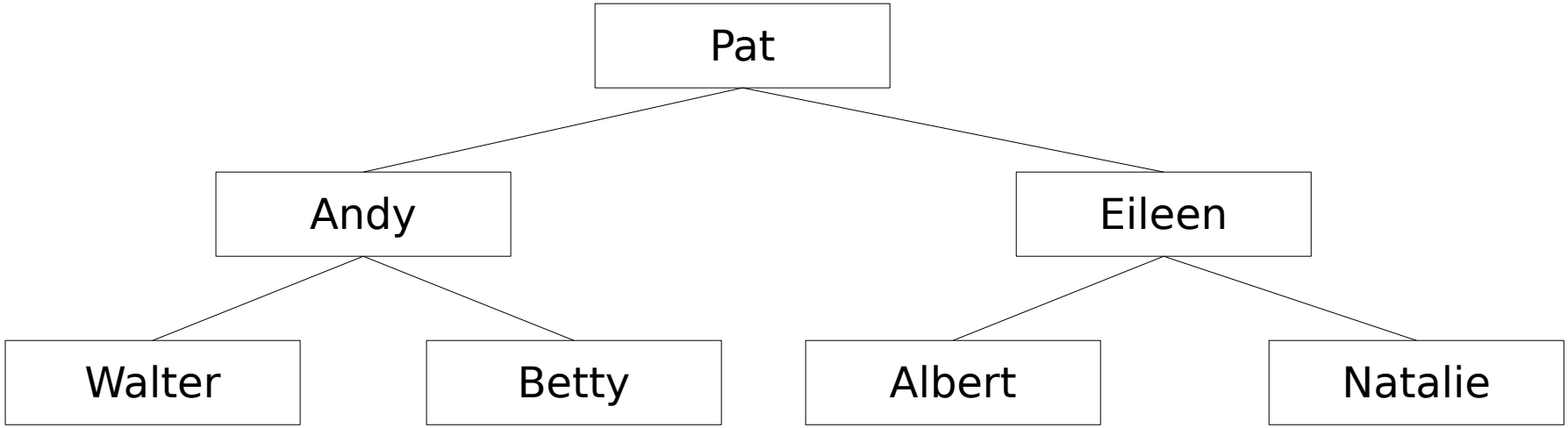
Hierarchical Organizations
Family Trees

- Family trees can be arranged in various ways.
- This diagram shows Emily's children and grandchildren.



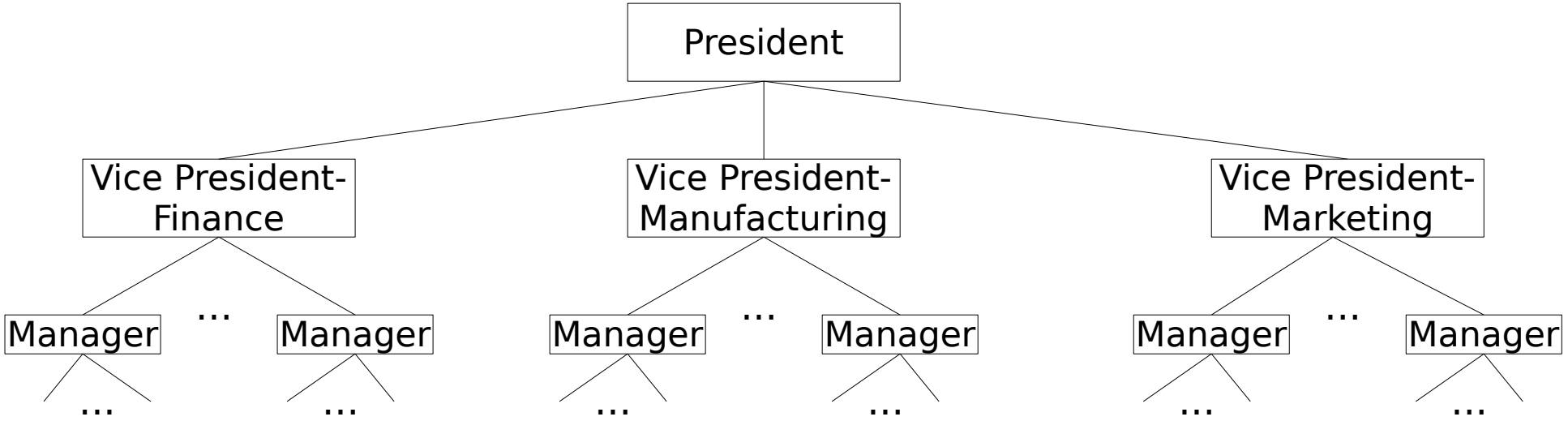
Hierarchical Organizations
Family Trees

- This diagram shows Pat's parents and grandparents.



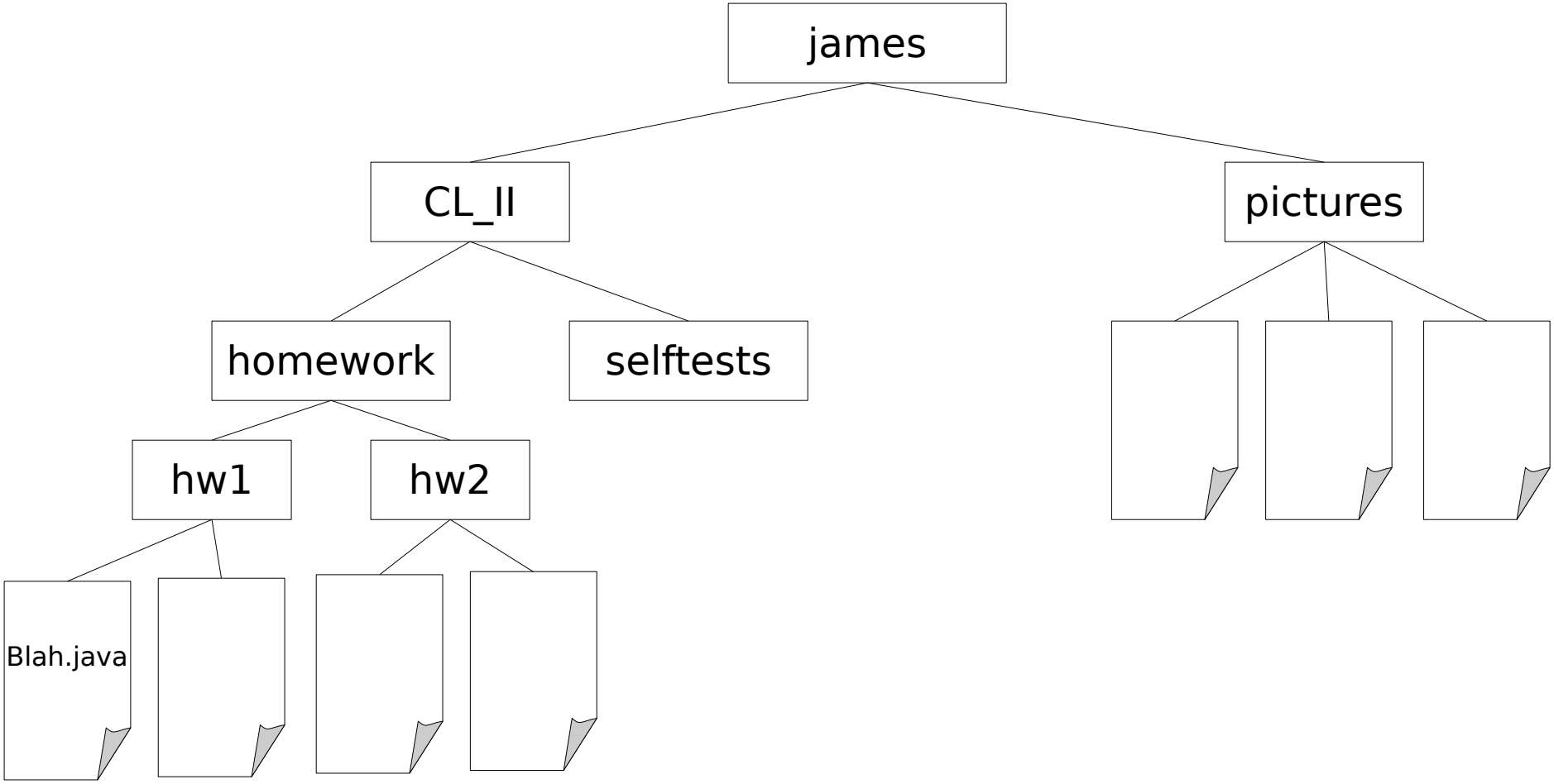
Hierarchical Organizations Businesses

- A hierarchical diagram of a business:

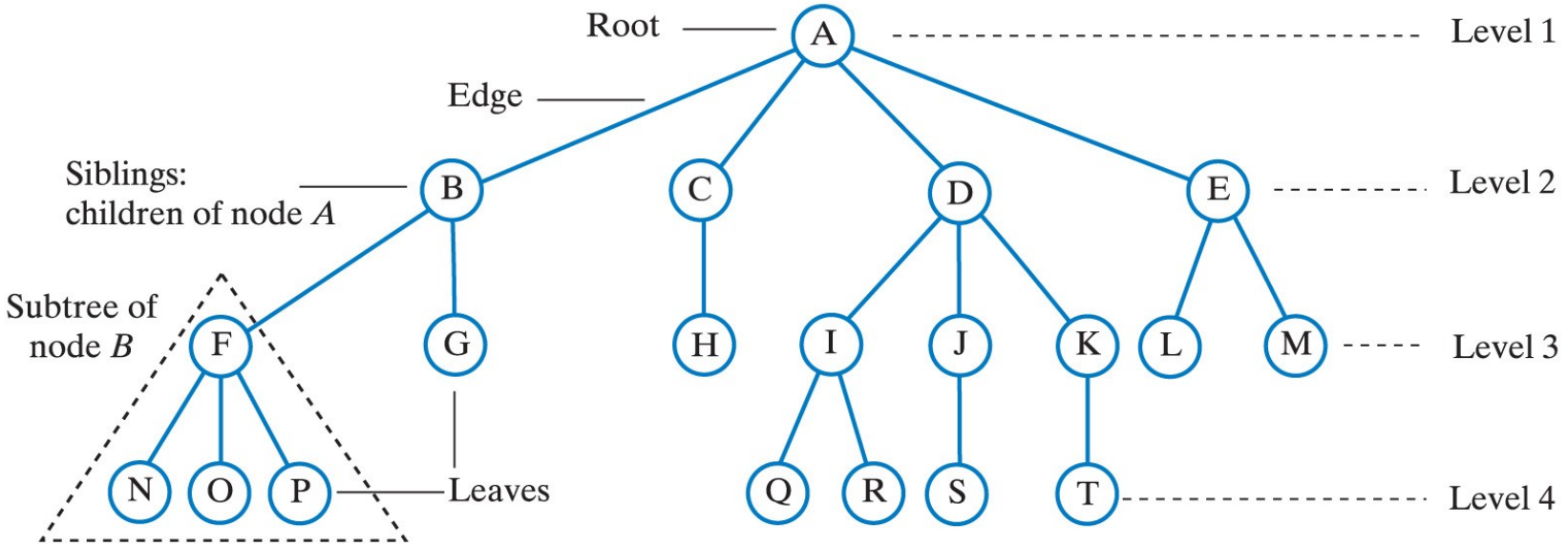


Hierarchical Organizations
Files and Directories

- Files and directories on a computer:

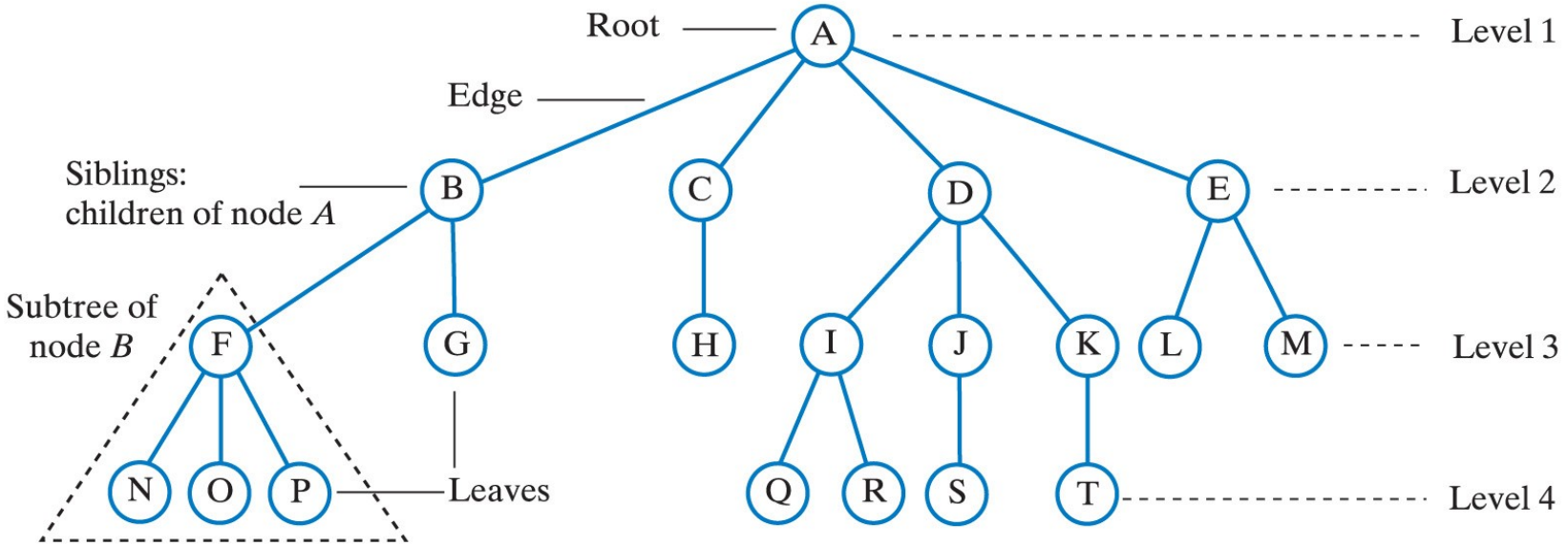


Terminology



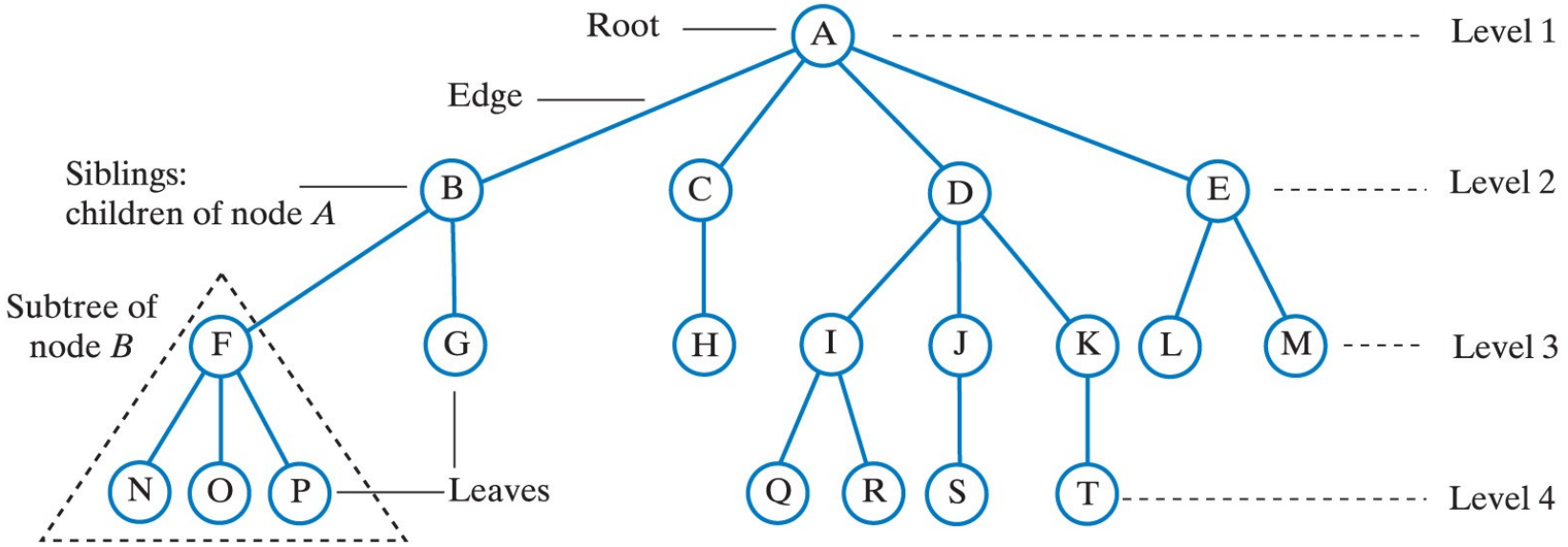
- A **tree** is a set of **nodes** connected by **edges** that show a relationship between the nodes.
- The nodes are arranged in levels that indicate the hierarchy of the nodes. The top level is a single node called the **root**.

Terminology



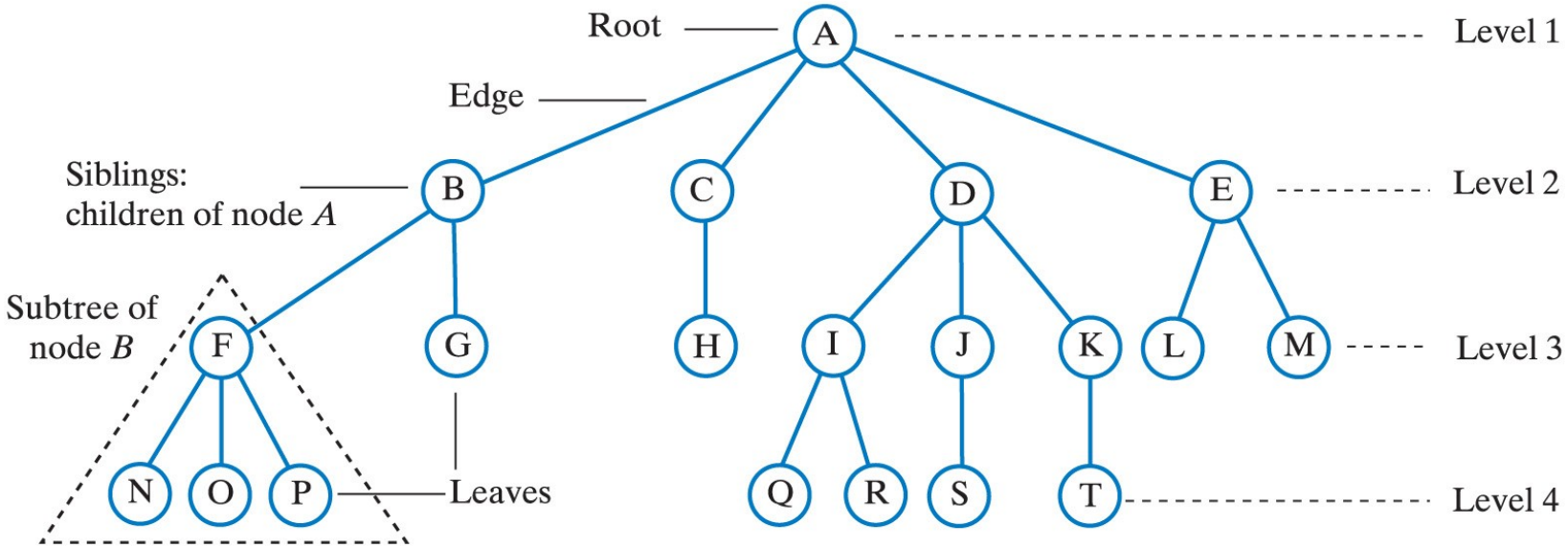
- The **children** of a node are those directly below it. *A* has 4 children: *B, C, D, E*
- The **parent** of a node is the node directly above it. *C*'s parent is *A*

Terminology



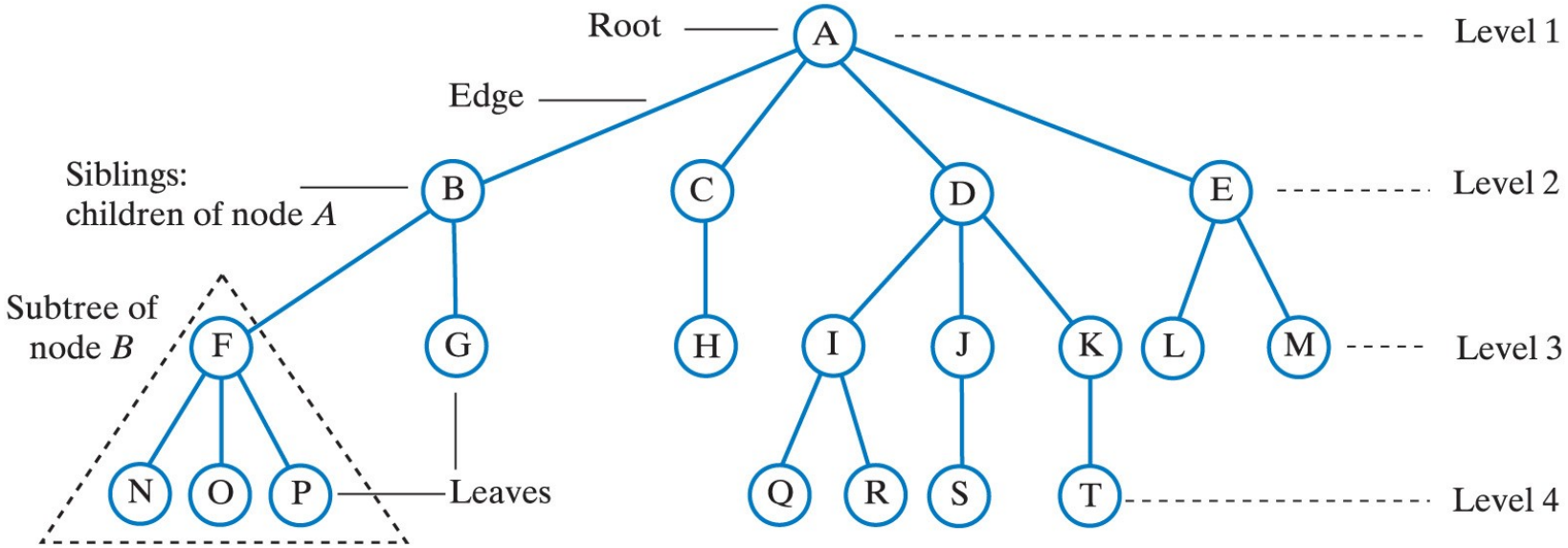
- All nodes have exactly 1 parent, except the root, which has no parent. *H*'s parent is *C*
- Nodes that share a parent are **siblings**. *B, C, D,* and *E* are siblings.

Terminology



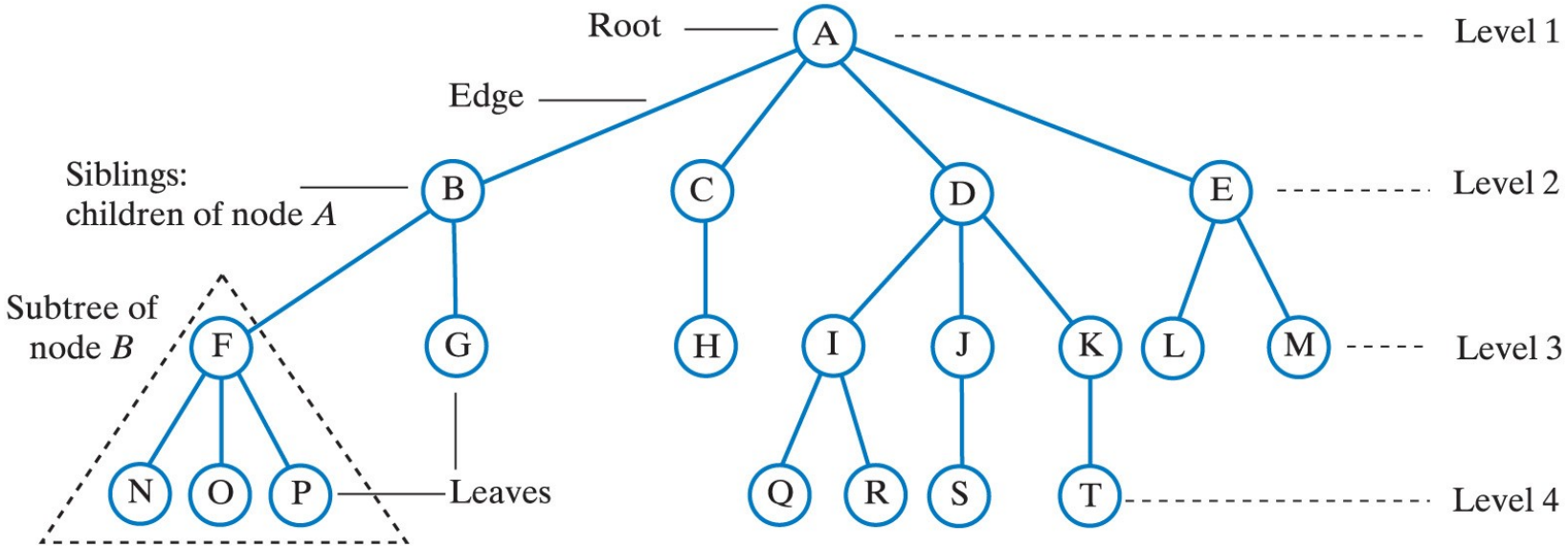
- The nodes below a given node (on the downward paths to the leaves) are its **descendants**.
- *D*'s descendants are *I, J, K, Q, R, S* and *T.*

Terminology



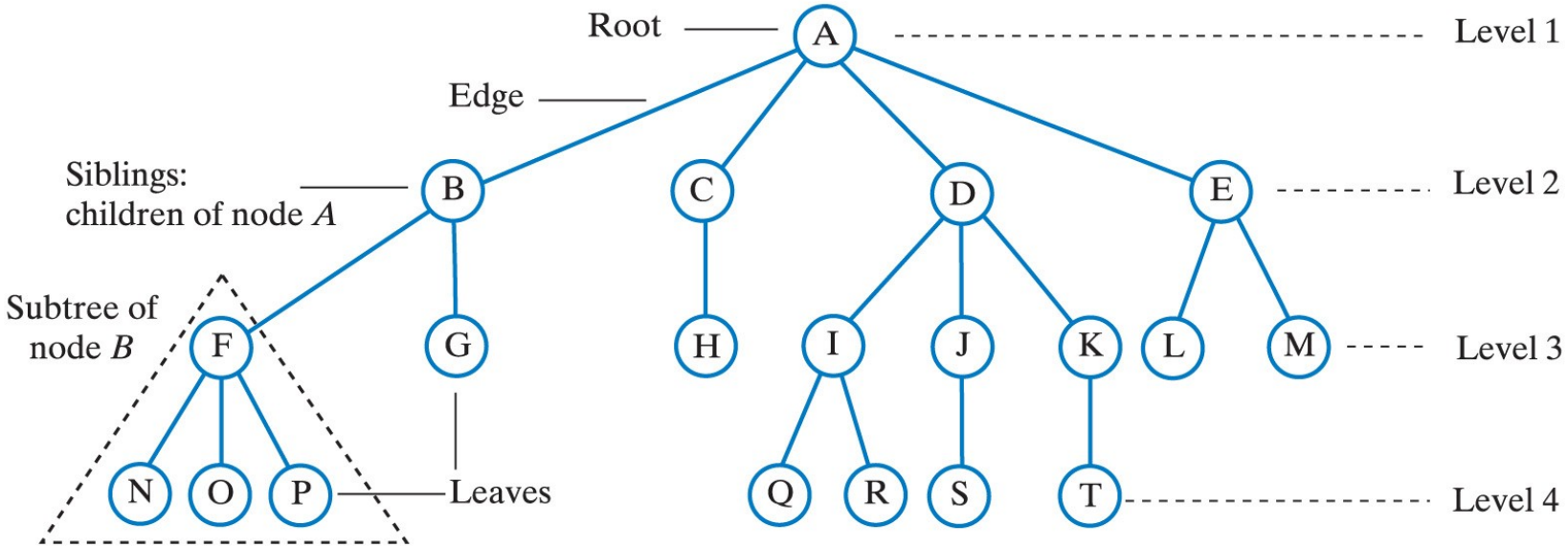
- The nodes <u>above</u> a given node (on the upward path towards the root) are its **ancestors**.
- *Q*'s ancestors are *I*, *D*, and *A*

Terminology



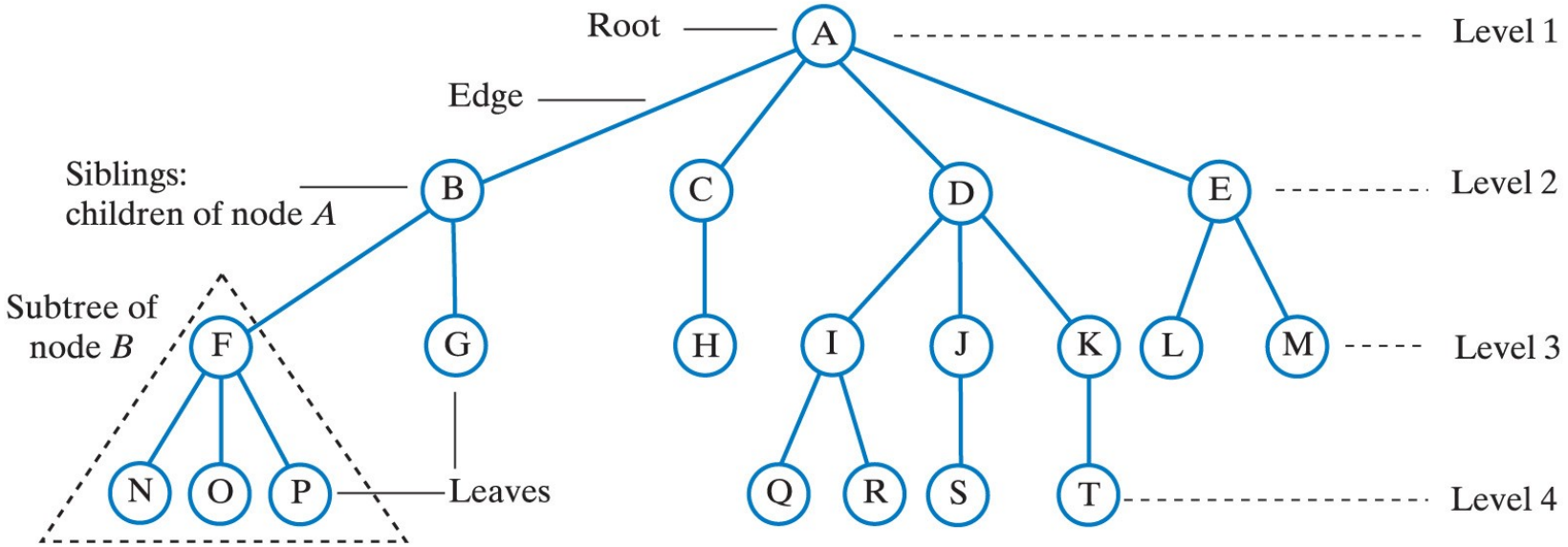
- A **leaf** is a node that has no children.
- Any node that is not a leaf is an **interior node** (or **non-leaf** node).

Exercise



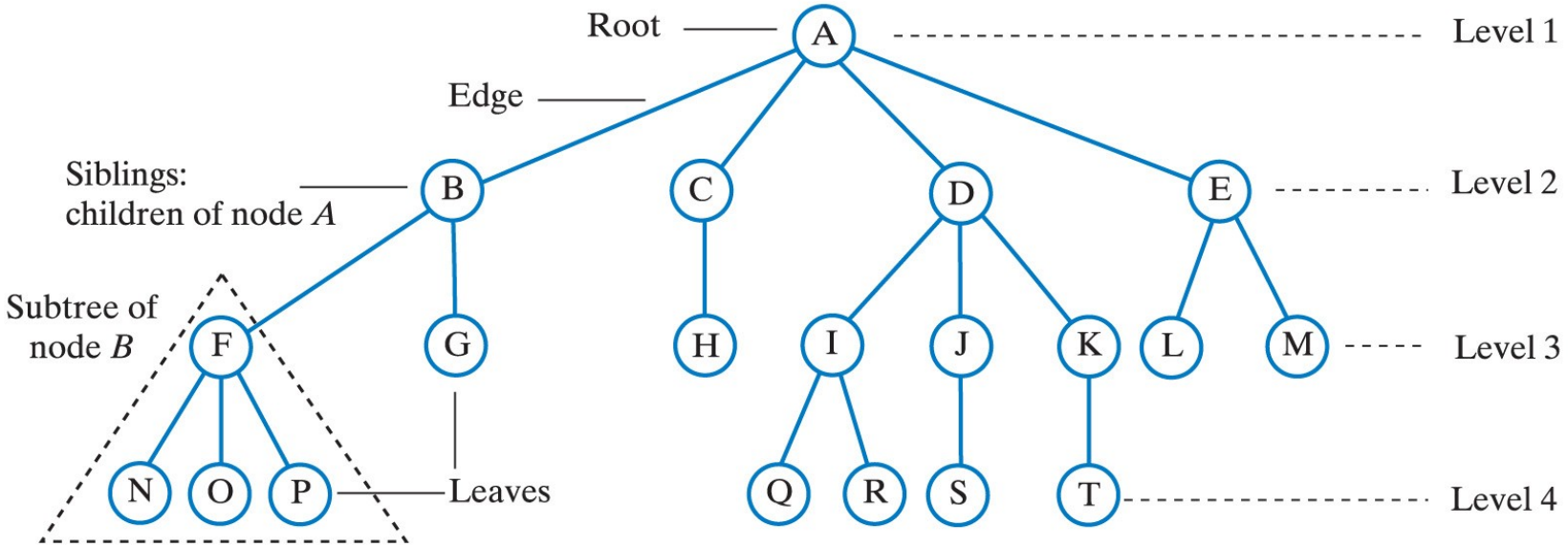
- What are the leaf nodes?
- What are the siblings of J?
- What are the children of E?
- What are the descendants of B?
- What are the ancestors of P?

Exercise



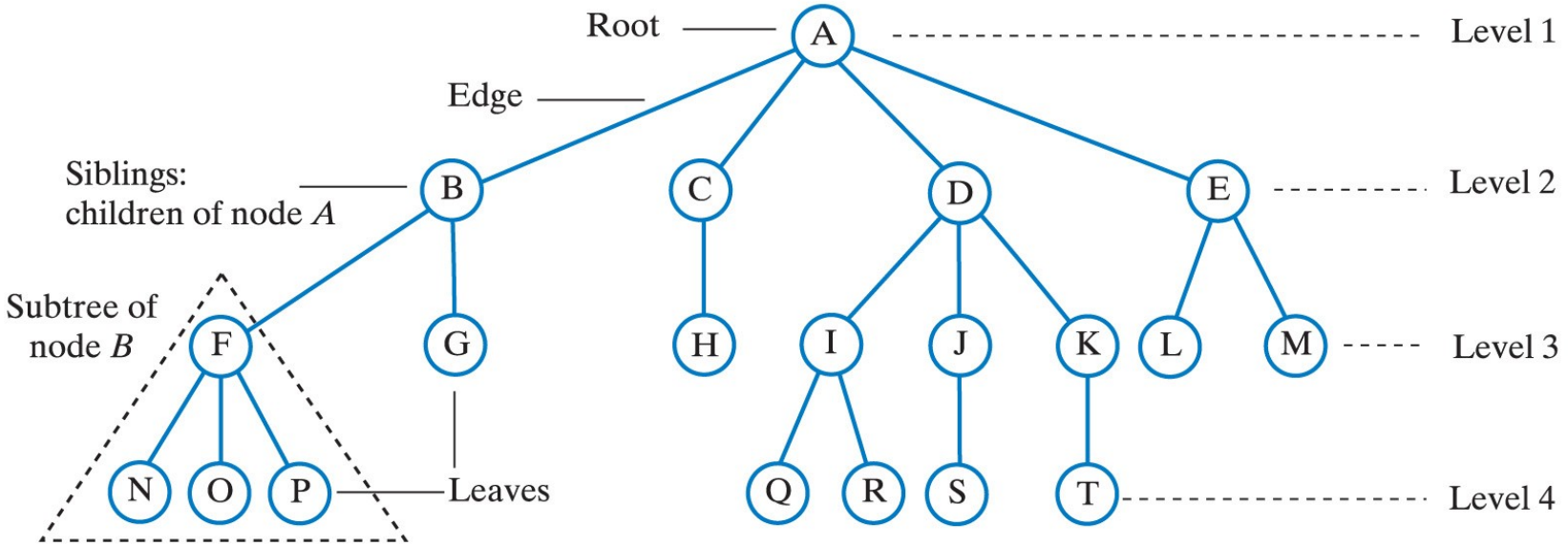
- What are the leaf nodes? N O P G H Q R S T L M
- What are the siblings of J?
- What are the children of E?
- What are the descendants of B?
- What are the ancestors of P?

Exercise



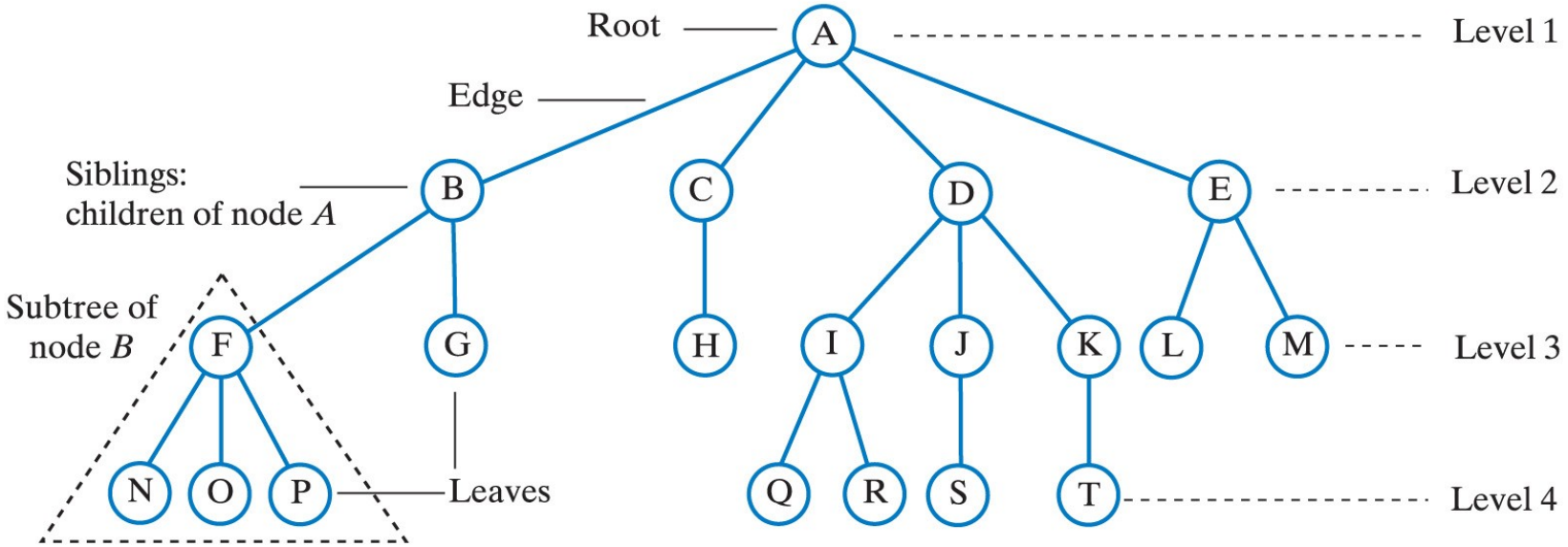
- What are the leaf nodes? N O P G H Q R S T L M
- What are the siblings of J? I K
- What are the children of E?
- What are the descendants of B?
- What are the ancestors of P?

Exercise



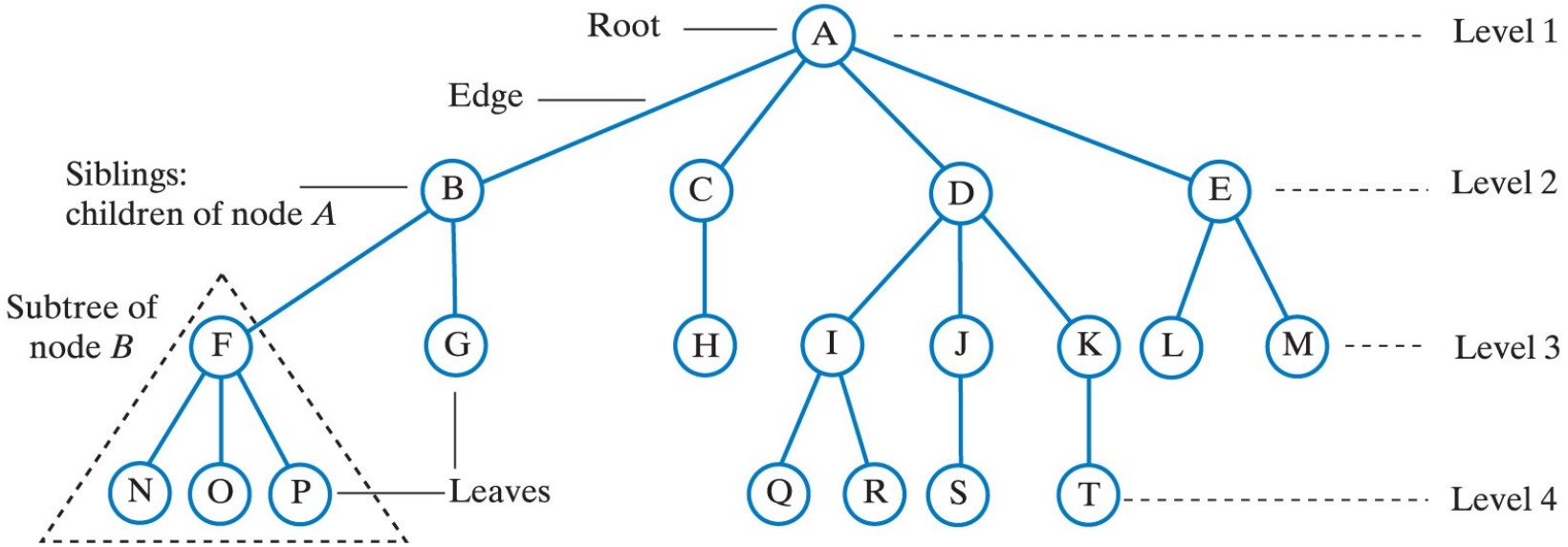
- What are the leaf nodes? N O P G H Q R S T L M
- What are the siblings of J? I K
- What are the children of E? L M
- What are the descendants of B?
- What are the ancestors of P?

Exercise



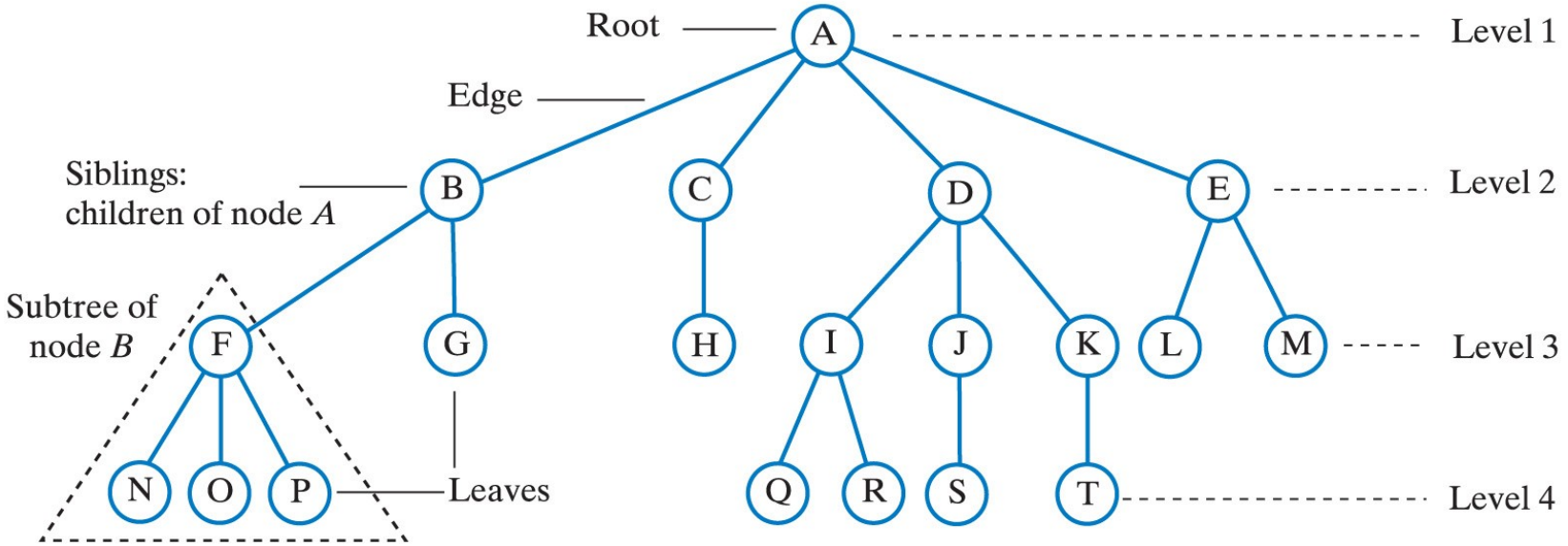
- What are the leaf nodes? N O P G H Q R S T L M
- What are the siblings of J? I K
- What are the children of E? L M
- What are the descendants of B? F G N O P
- What are the ancestors of P?

Exercise



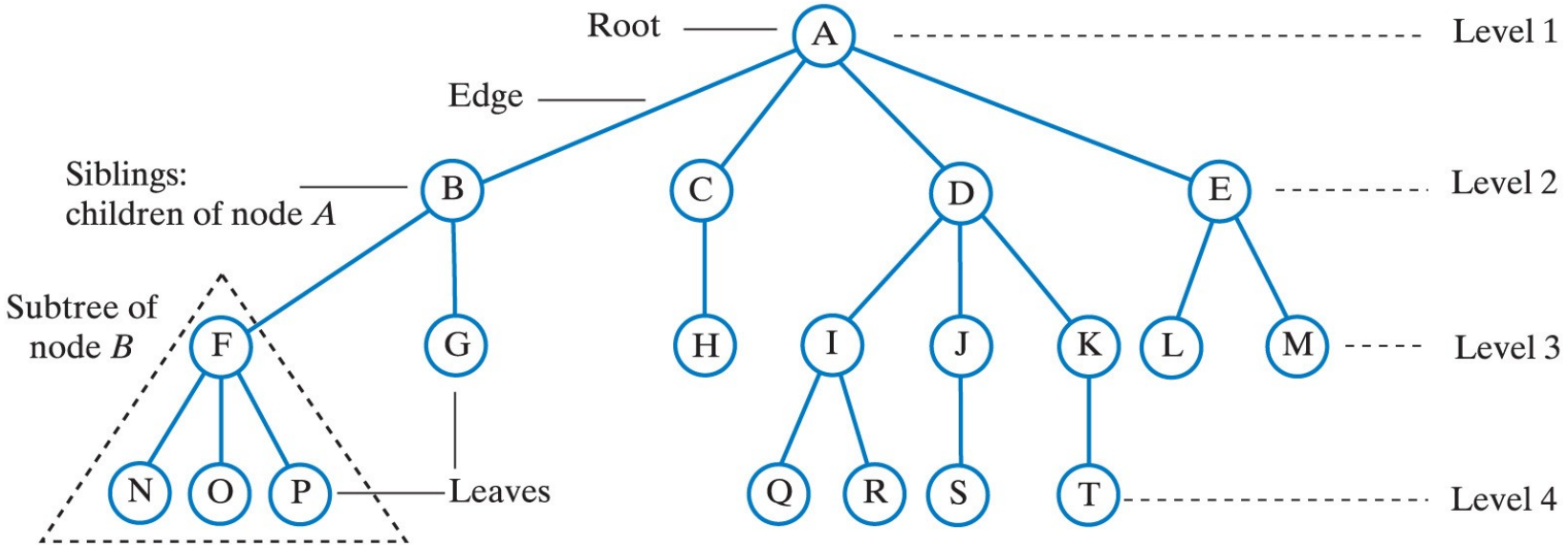
- What are the leaf nodes? N O P G H Q R S T L M
- What are the siblings of J? I K
- What are the children of E? L M
- What are the descendants of B? F G N O P
- What are the ancestors of P? F B A

More Terminology



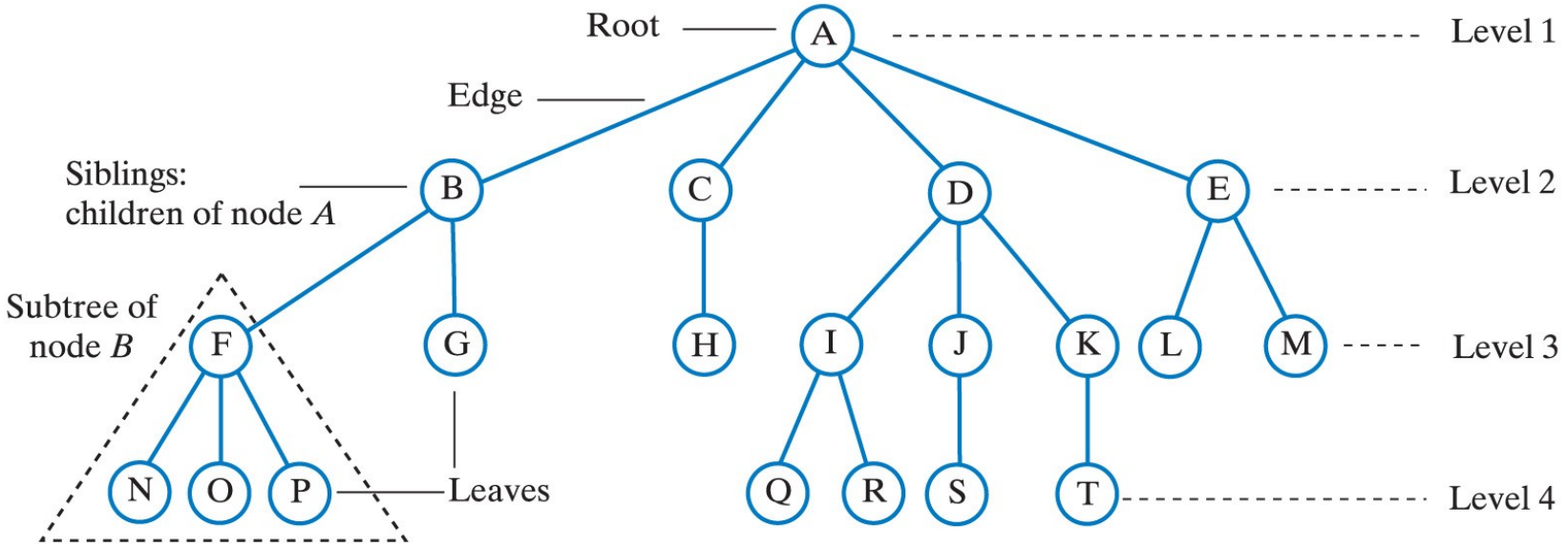
- A **subtree of a node** is a tree rooted at one of that node's children.
- Node *B* has 2 subtrees.
- Node *A* has 4 subtrees.

More Terminology



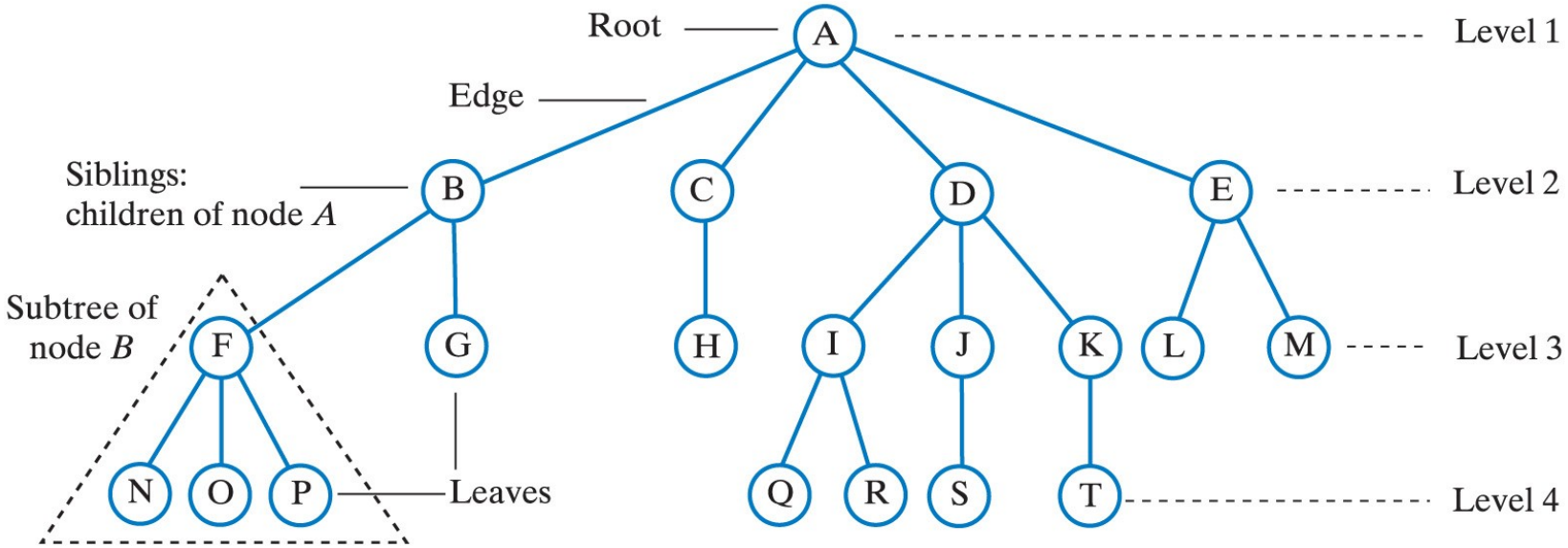
- We can reach any node in a tree by following a **path** that begins at the root and goes from node to node along the edges that connect them.
- The path to *R* is *A D I R*

More Terminology



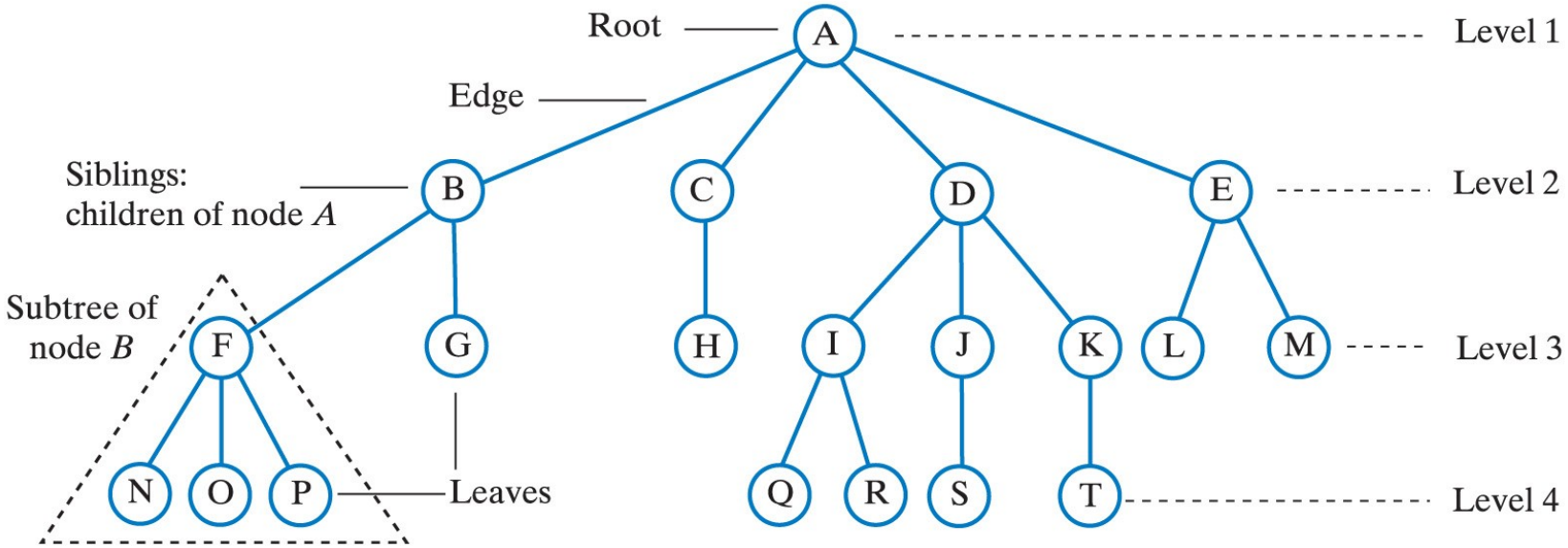
- The **length of a path** is the number of edges that compose it.
- The length of the path to *R* is 3

More Terminology



- The **height** of a tree is the number of levels in the tree, or the number of nodes along the longest path between the root and a leaf.
- This example tree has height 4.

More Terminology



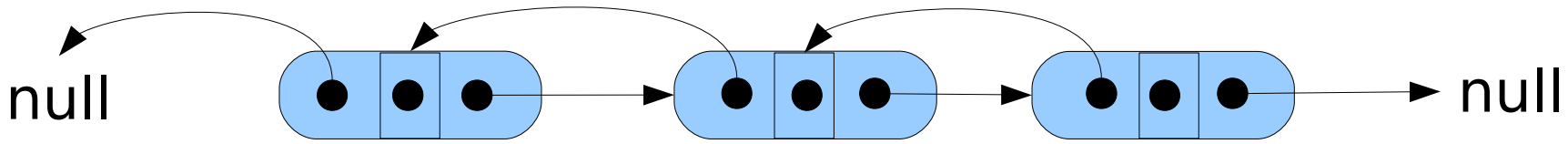
- The path between the root and any other node is unique – there is no circularity in a tree.
- A tree-like data structure that has circularity is called a **graph**.

More Terminology

- In a **general tree**, each node can have any number of children.
- A tree in which the nodes have at most *n* children is called an **n-ary** tree.
- In particular, a tree whose nodes have at most 2 children is called a **binary tree**.

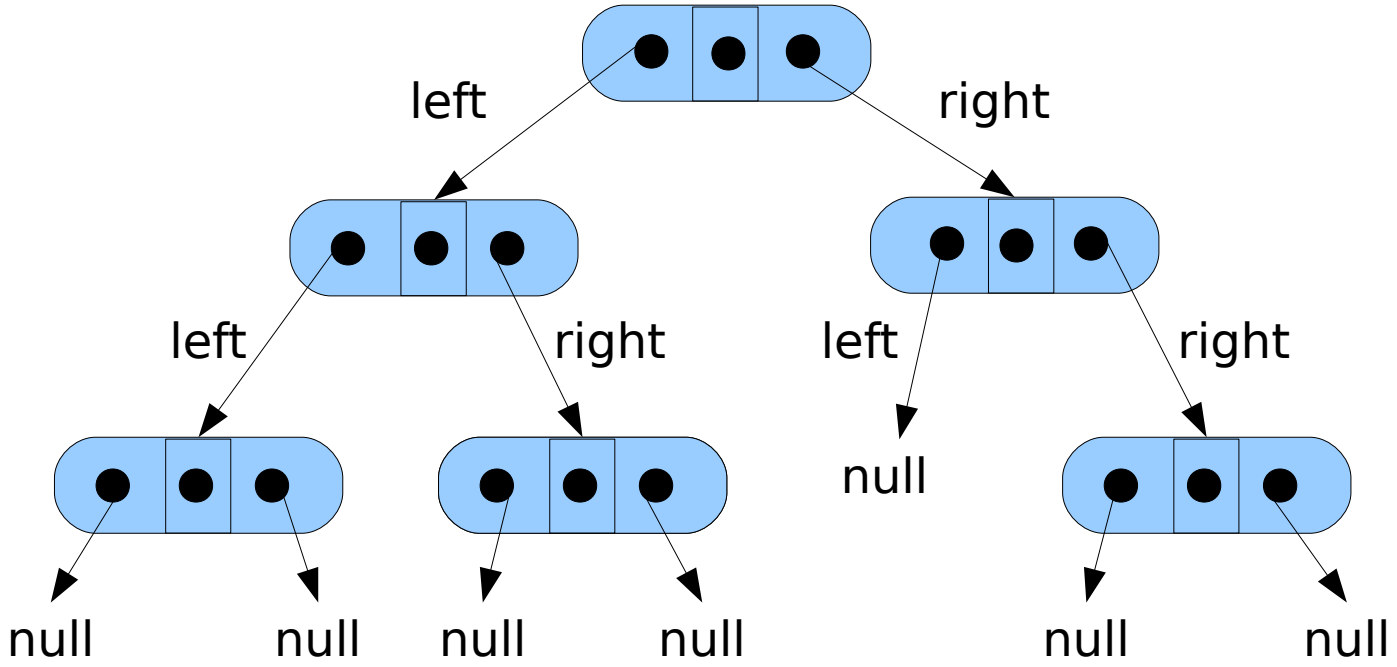
Binary Trees

- We know how to create and link nodes together to form a list.
- A doubly-linked list can be formed by using two references in the node object – one to point to the next node and one to point to the previous node in the list.



Binary Trees

- By using a similar node class with 2 references, we can form binary trees.
- Each node contains a reference to its *left* and its *right* subtree.



Tree Traversals

- When we iterate a linear data structure, such as a list, the order in which to process the data is clear.
- When we iterate a tree it is called a **traversal**, and the order of processing the nodes is not unique.
- Each node must be **visited** (i.e. processed in some way) exactly once.

Tree Traversals

- We know that the subtrees of the root of a binary tree are also binary trees.
- We can use the recursive nature of a binary tree to define its traversal.
- To visit all the nodes in a binary tree we have to
  - visit the root
  - visit all nodes in the left subtree
  - visit all nodes in the right subtree

Tree Traversals

- Visiting the left subtree before the right subtree is simply a convention.
- The root can be visited before, between, or after its two subtrees.

root
left subtree
right subtree

left subtree
root
right subtree

left subtree
right subtree
root

Tree Traversals

- The order in which the root node and its subtrees are visited allows us to define the 3 most common tree traversals
  - preorder
  - inorder
  - postorder
- The *pre-*, *in-*, and *post-* refer to the visitation of the root node.

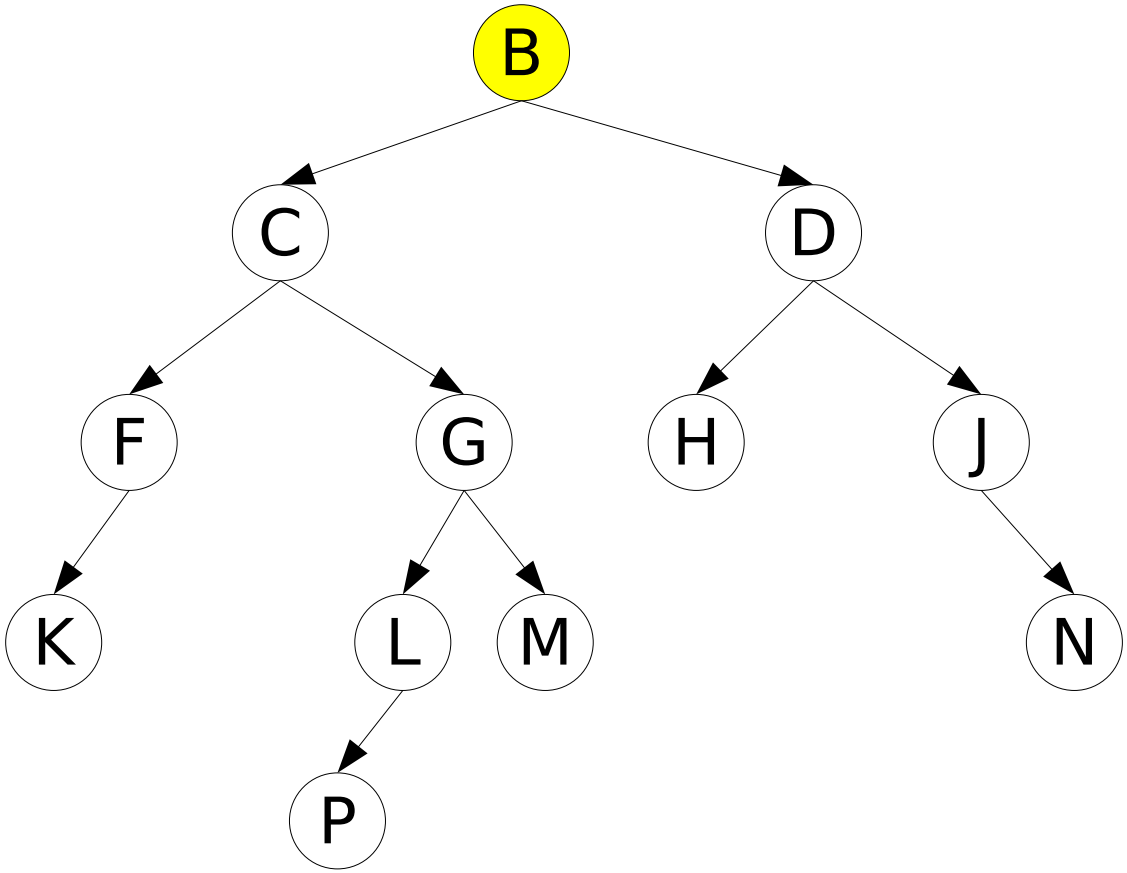
pre-order	**in-order**	**post-order**
root	left subtree	left subtree
left subtree	**root**	right subtree
right subtree	right subtree	**root**

Preorder Traversal

- In a preorder traversal, the root node is visited before its subtrees:
  - **visit the root**
  - visit all nodes in the left subtree
  - visit all nodes in the right subtree

Preorder Traversal Example

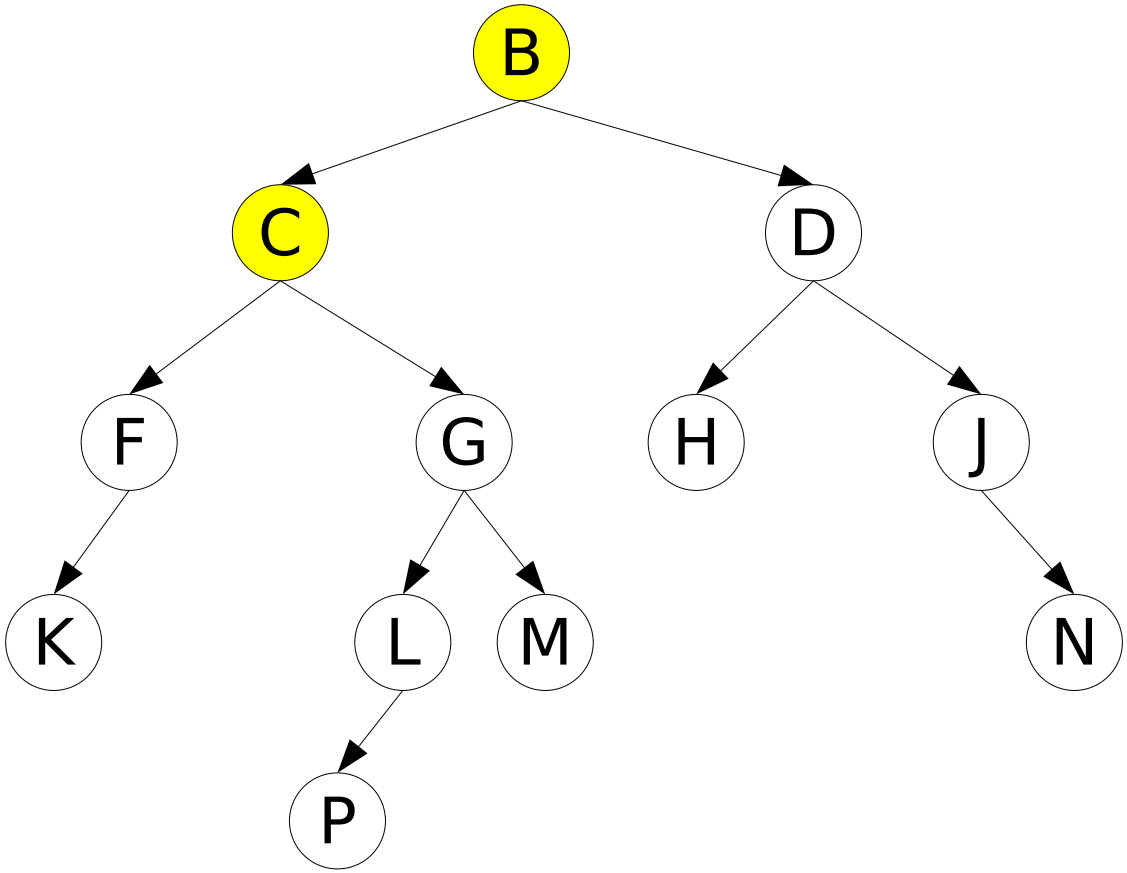
root
left
right



B

Preorder Traversal Example

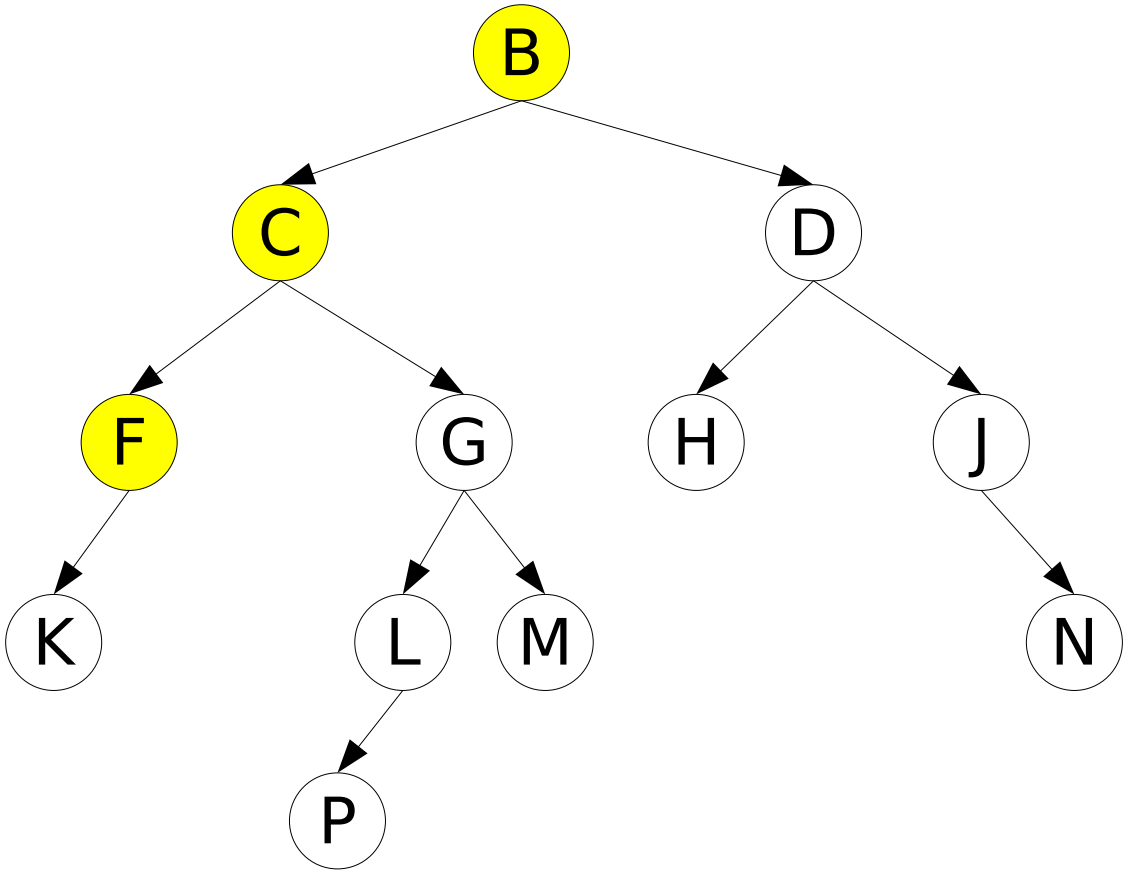
root
left
right



B C

Preorder Traversal Example

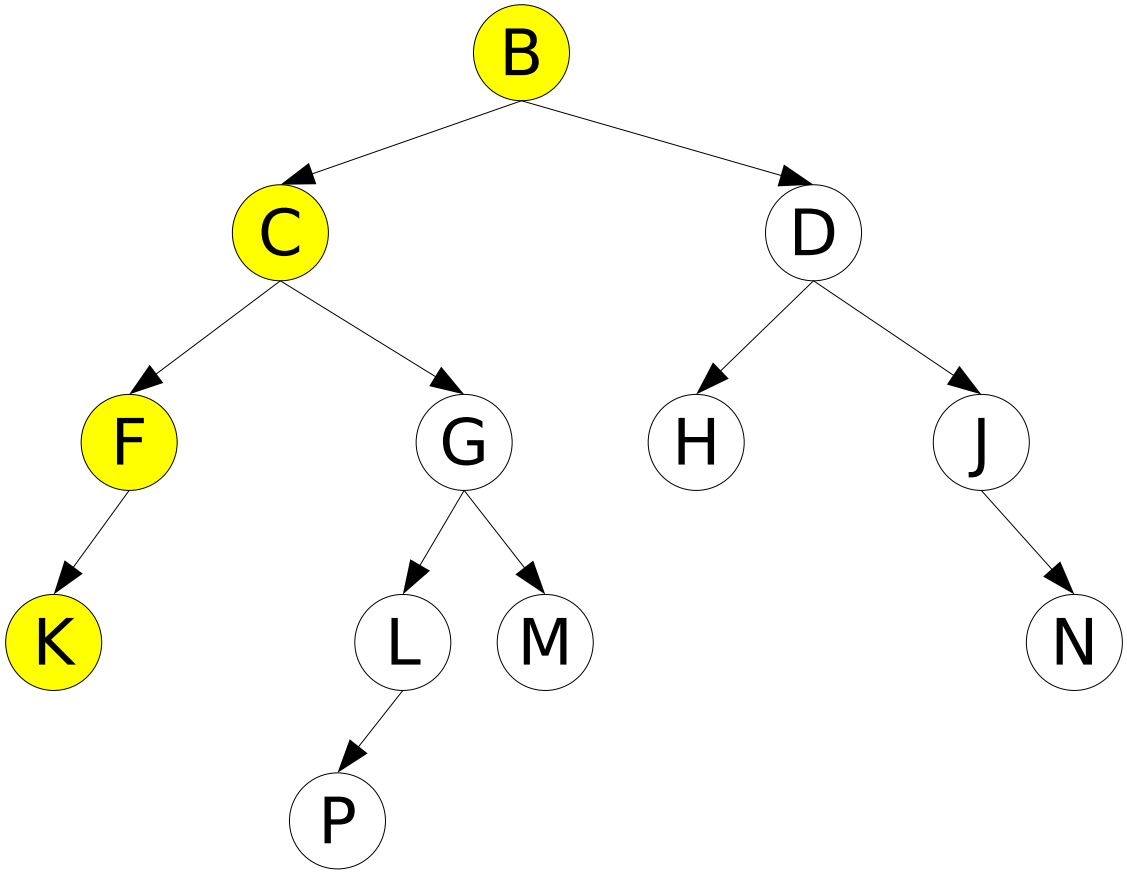
root
left
right



B C F

Preorder Traversal Example

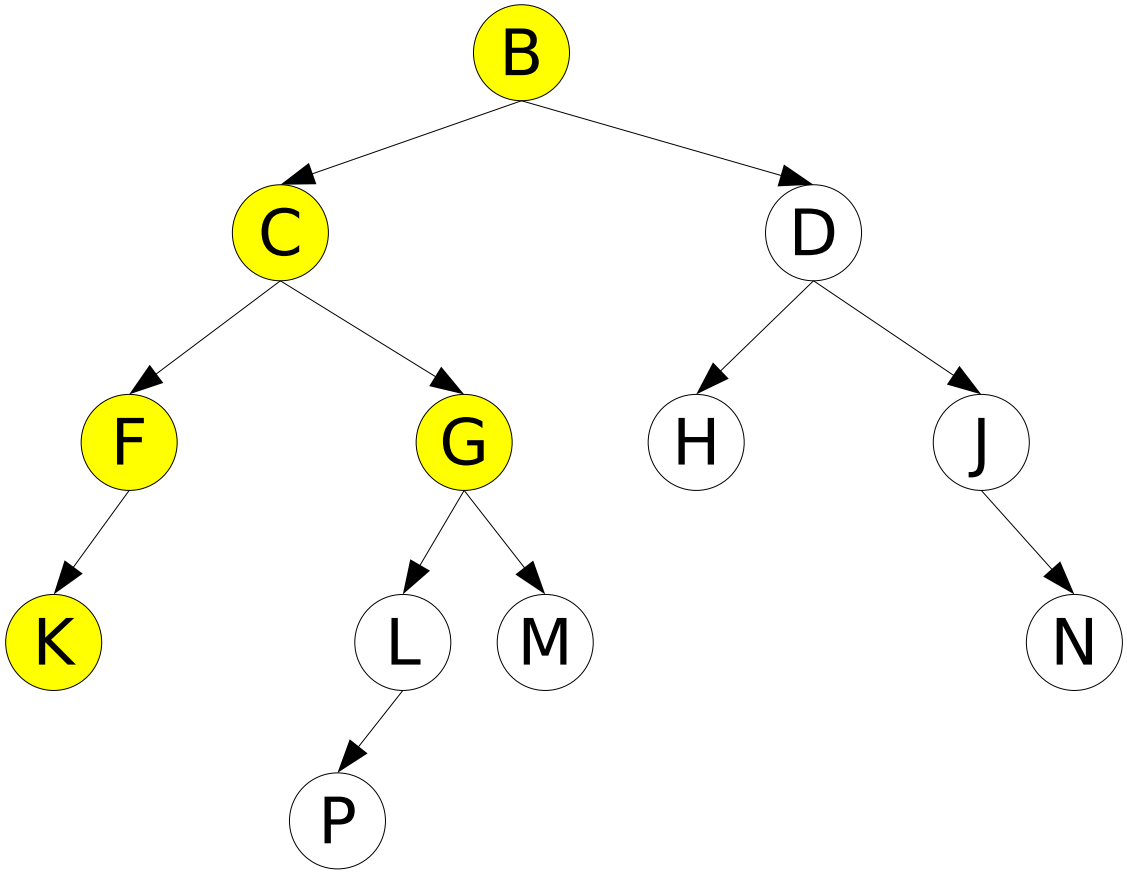
root
left
right



B C F K

Preorder Traversal Example

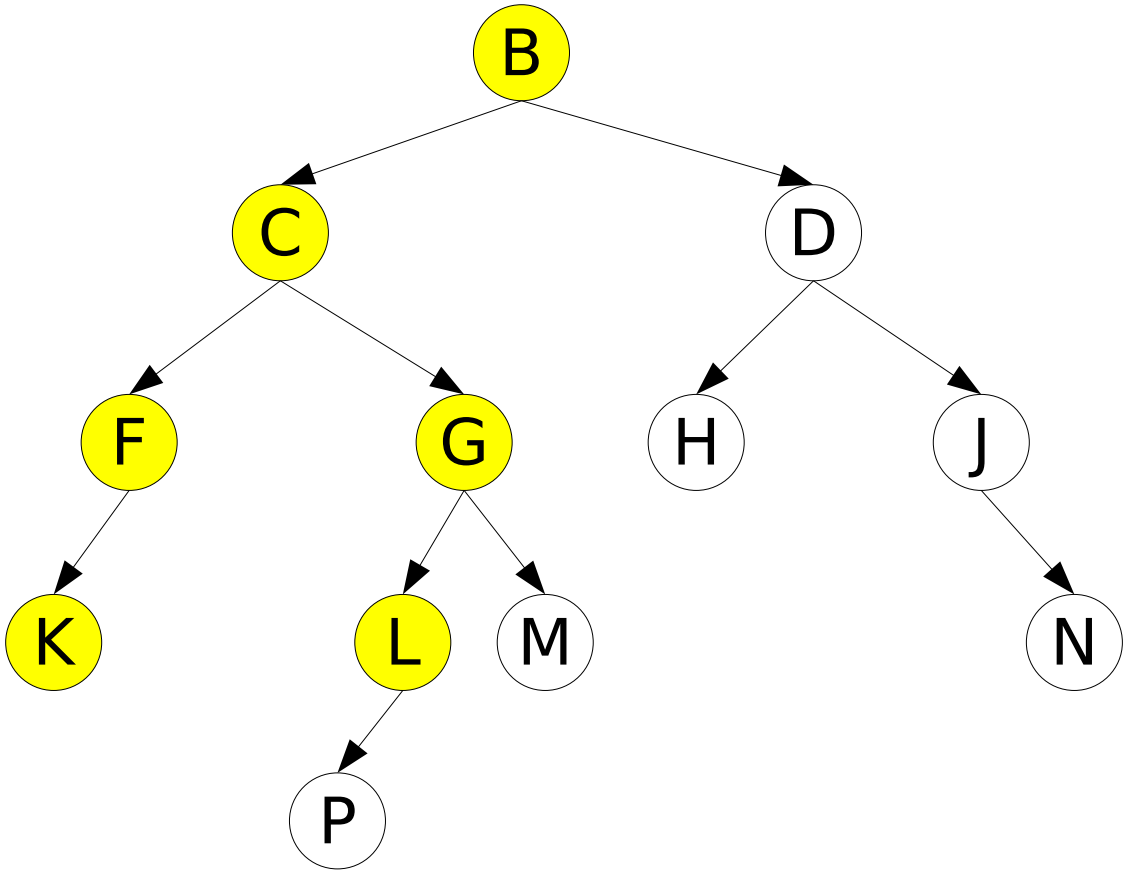
root
left
right



B C F K G

Preorder Traversal Example

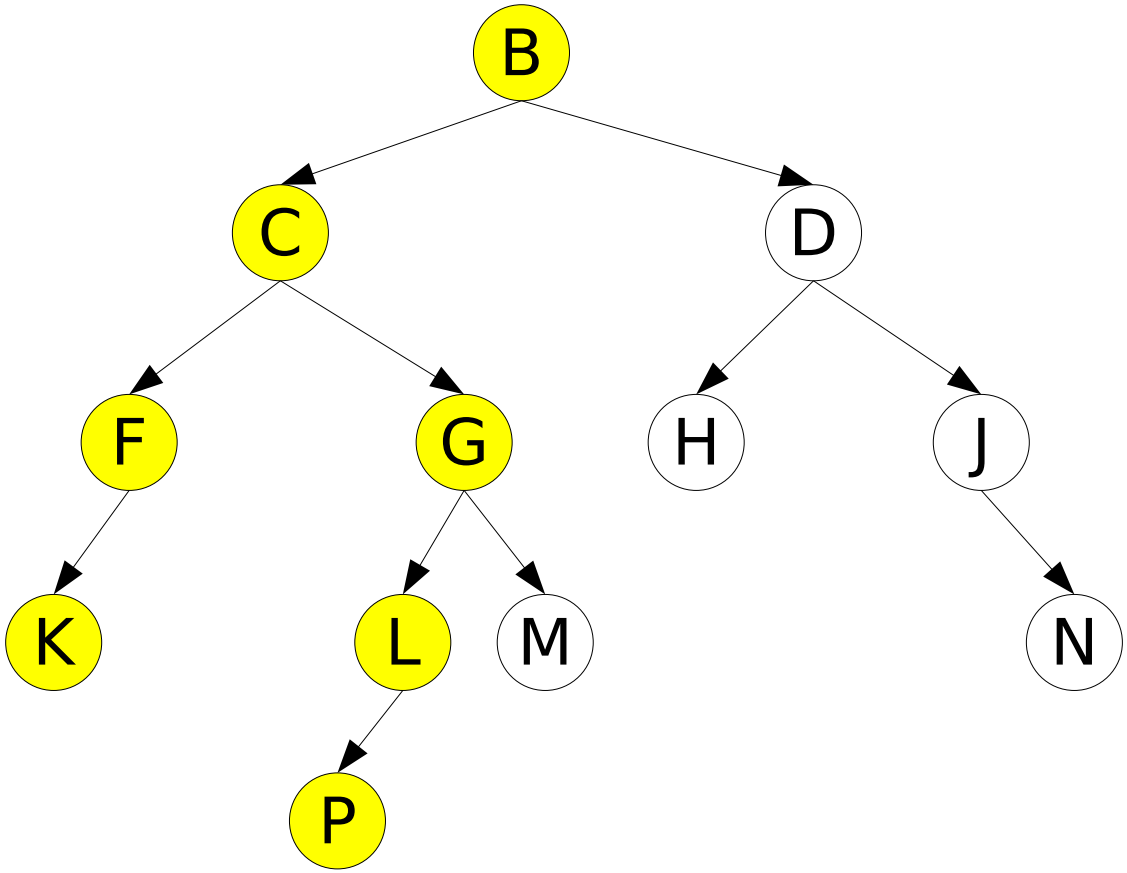
root
left
right



B C F K G L

Preorder Traversal Example

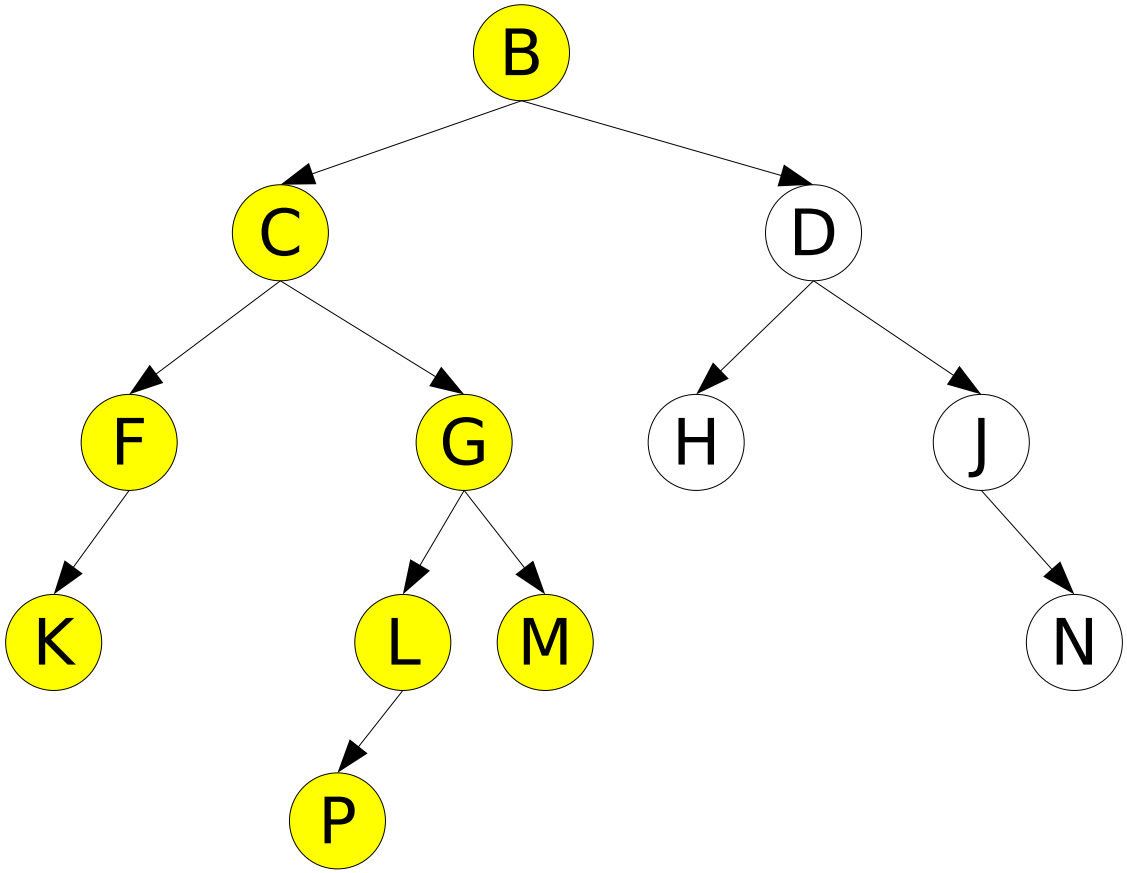
root
left
right



B C F K G L P

Preorder Traversal Example

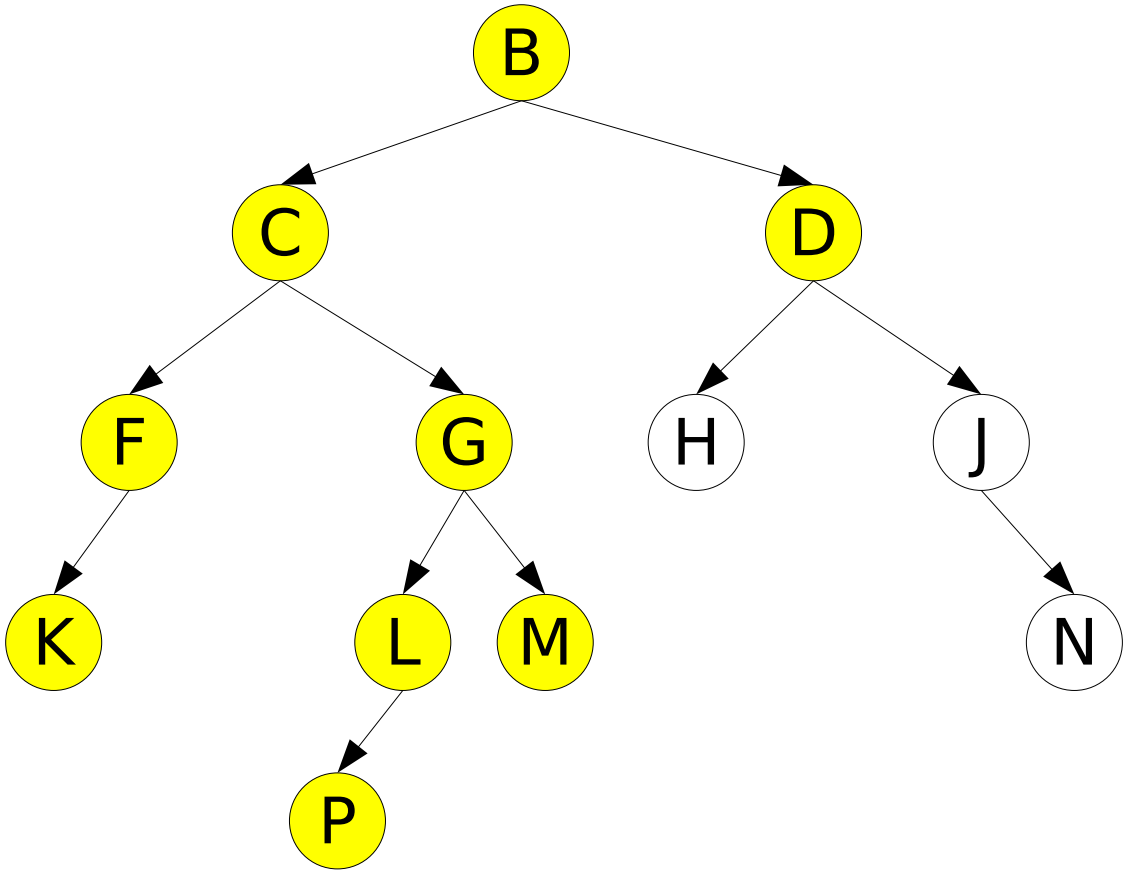
root
left
right



B C F K G L P M

Preorder Traversal Example

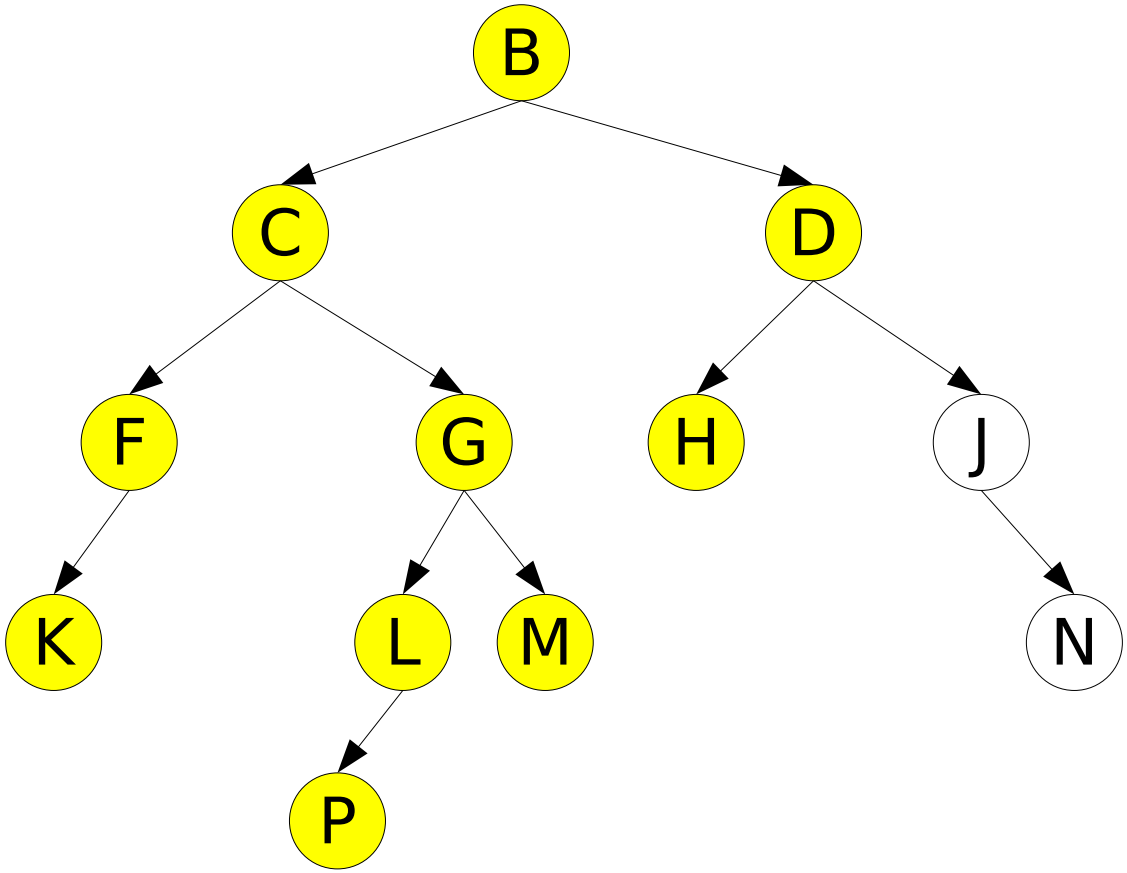
root
left
right



B C F K G L P M D

Preorder Traversal Example

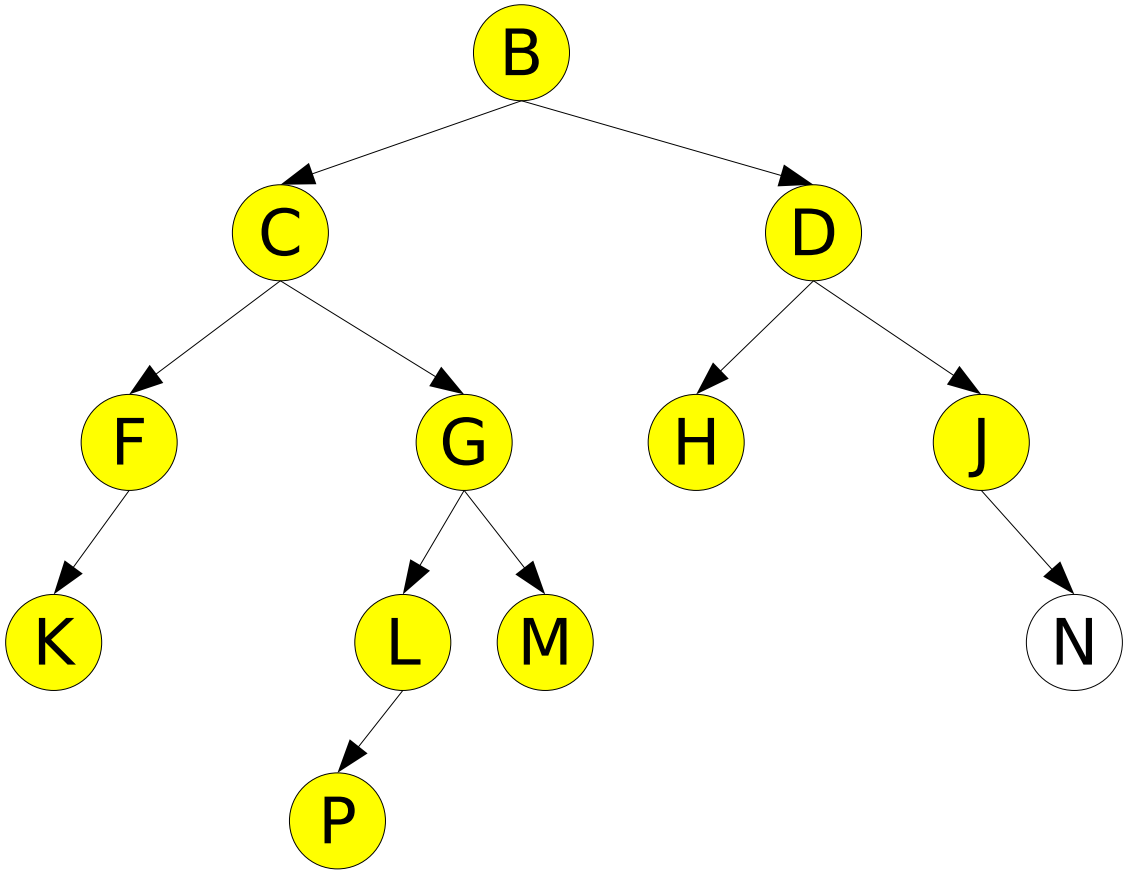
root
left
right



B C F K G L P M D H

Preorder Traversal Example

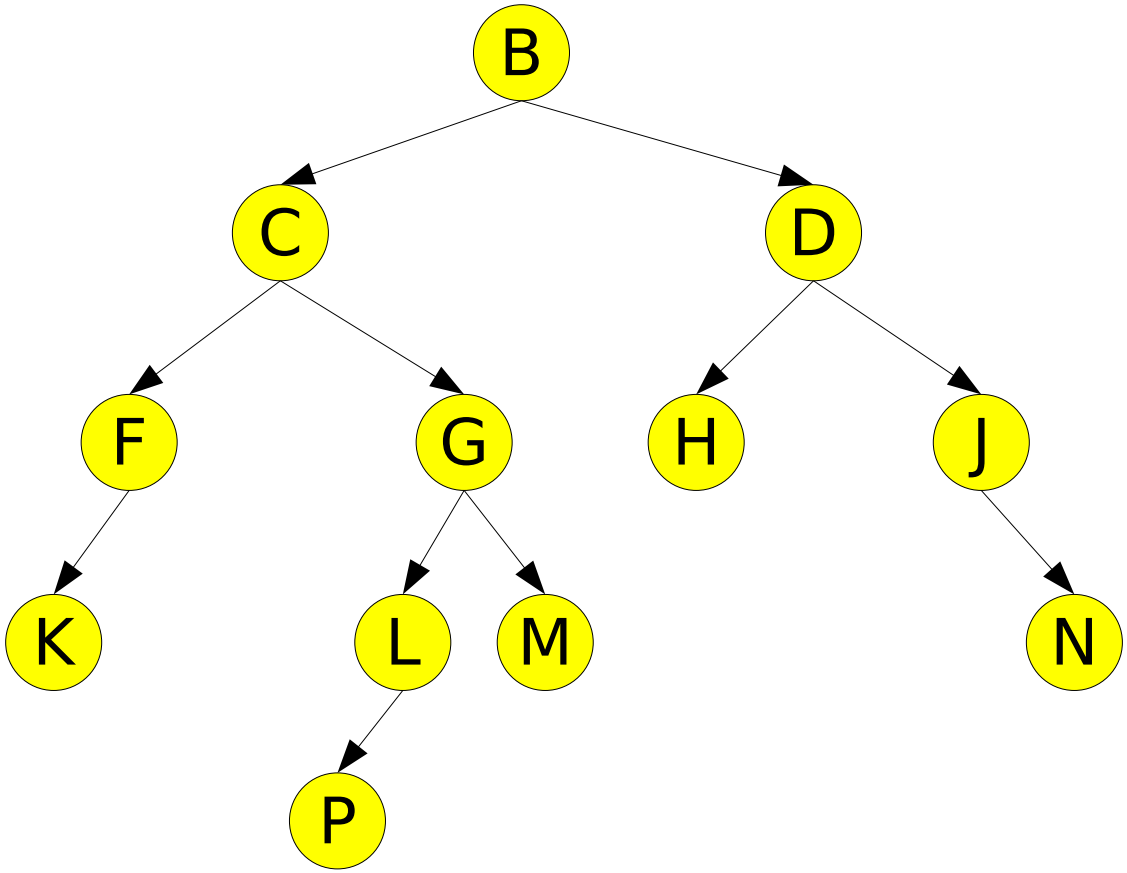
root
left
right



B C F K G L P M D H J

Preorder Traversal Example

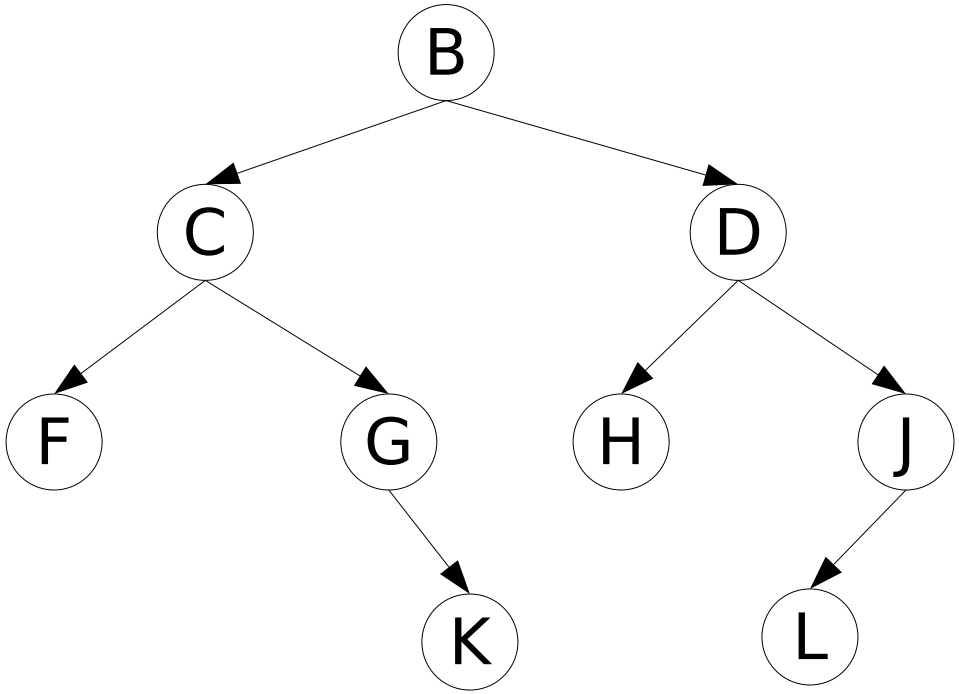
root
left
right



B C F K G L P M D H J N

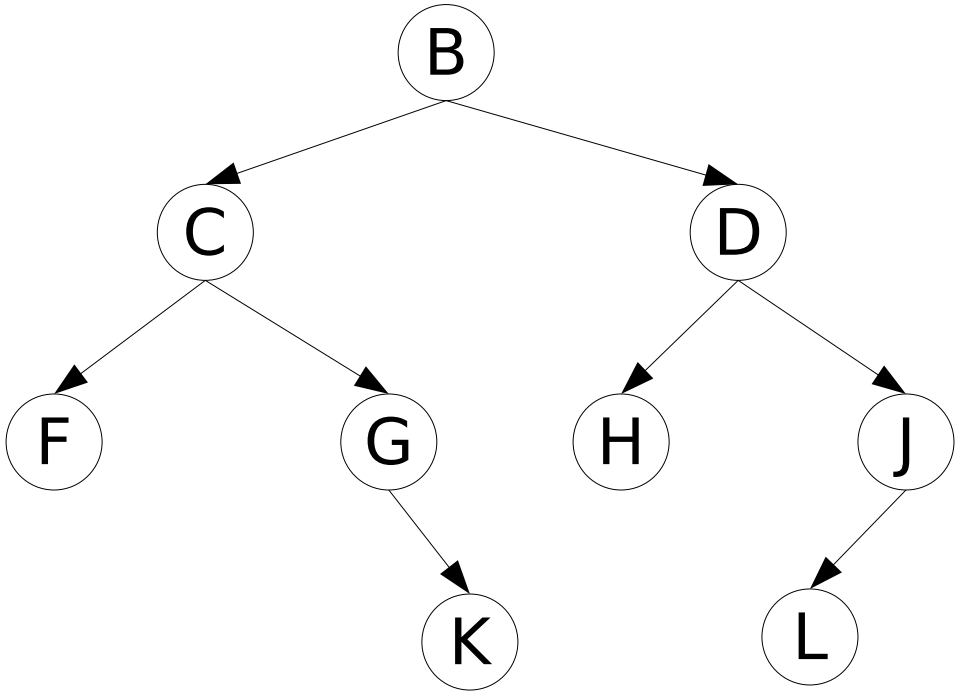
Preorder Traversal Exercise

What is the preorder traversal of this tree?



Preorder Traversal Exercise

What is the preorder traversal of this tree?



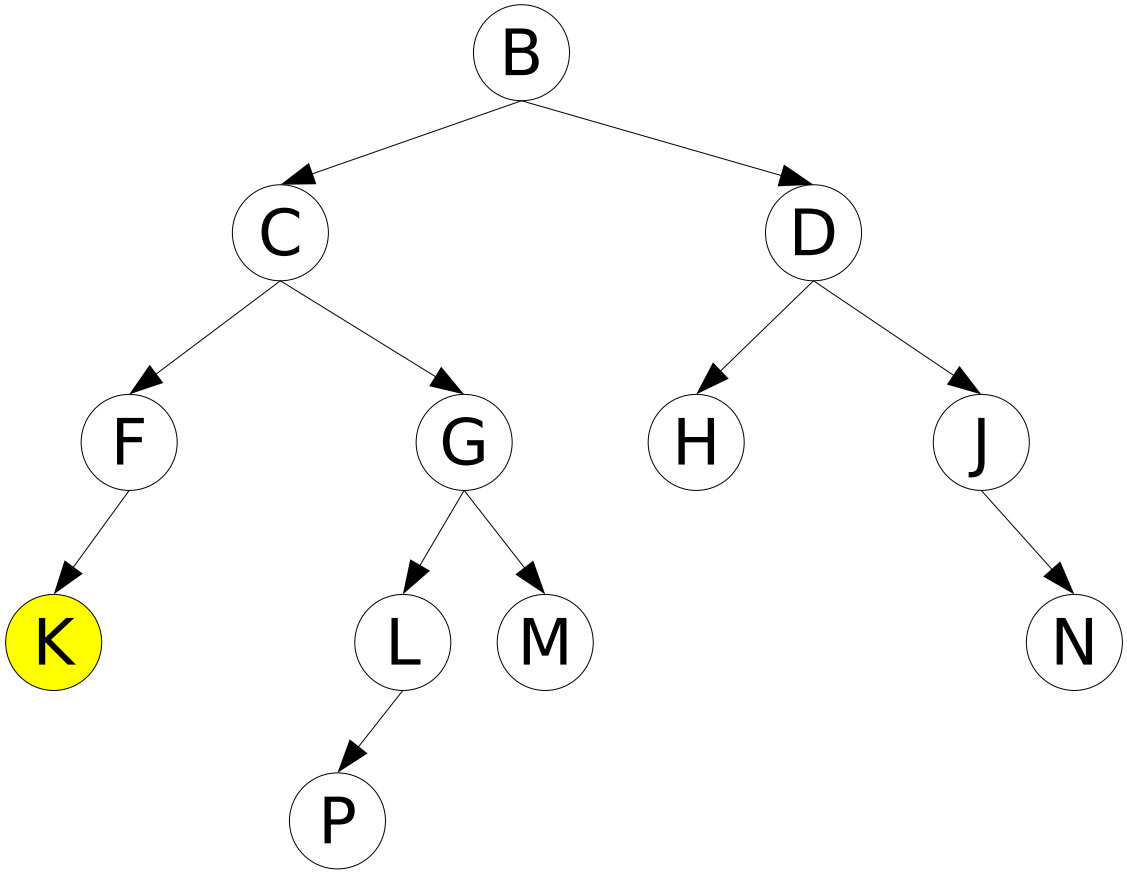
B C F G K D H J L

Inorder Traversal

- In an inorder traversal, the root node is visited between its subtrees:
  - visit all nodes in the left subtree
  - **visit the root**
  - visit all nodes in the right subtree

Inorder Traversal Example

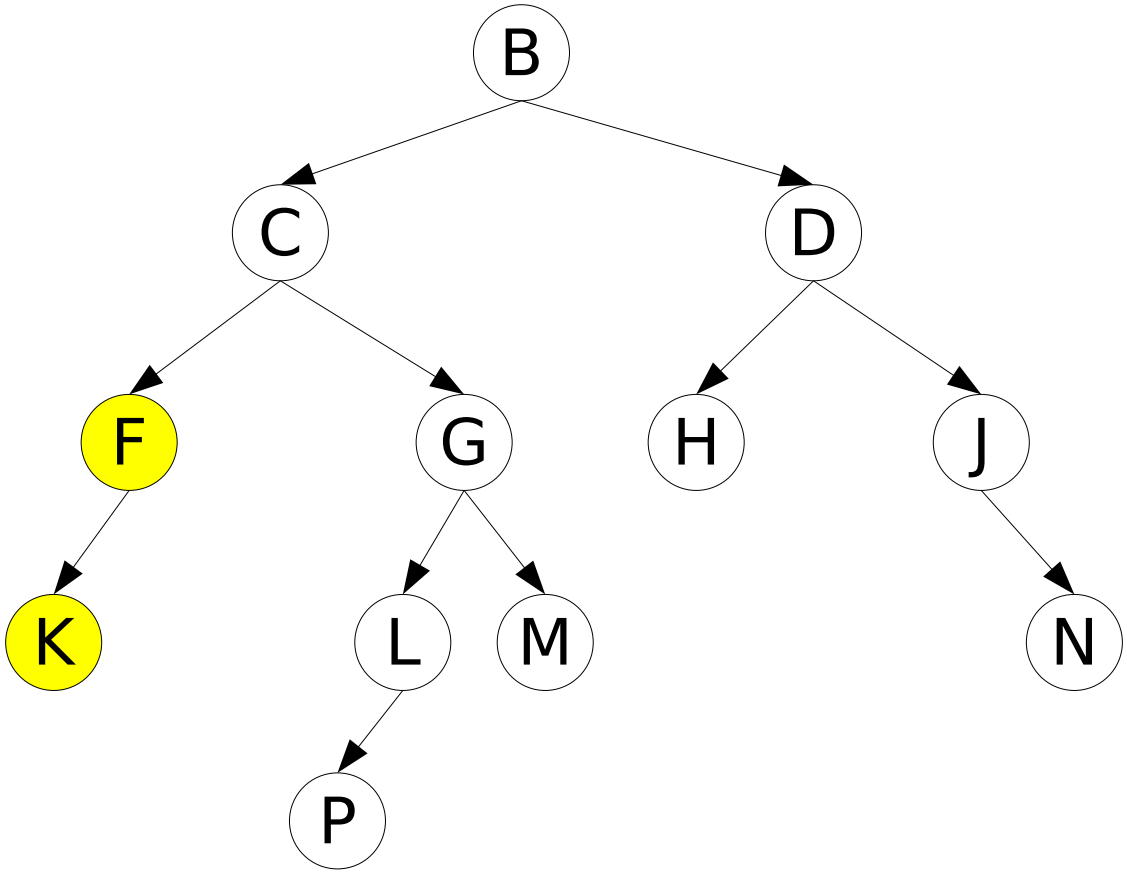
left
root
right



K

Inorder Traversal Example

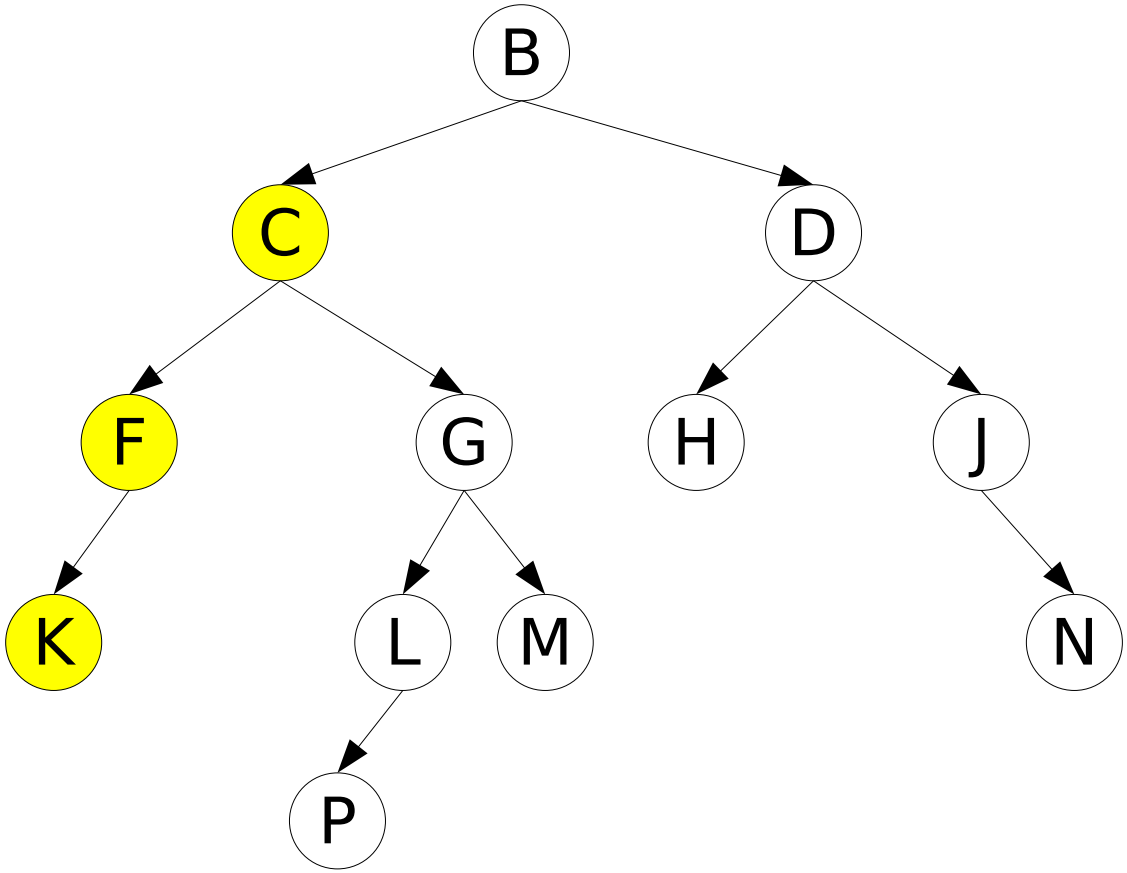
left
root
right



K F

Inorder Traversal Example

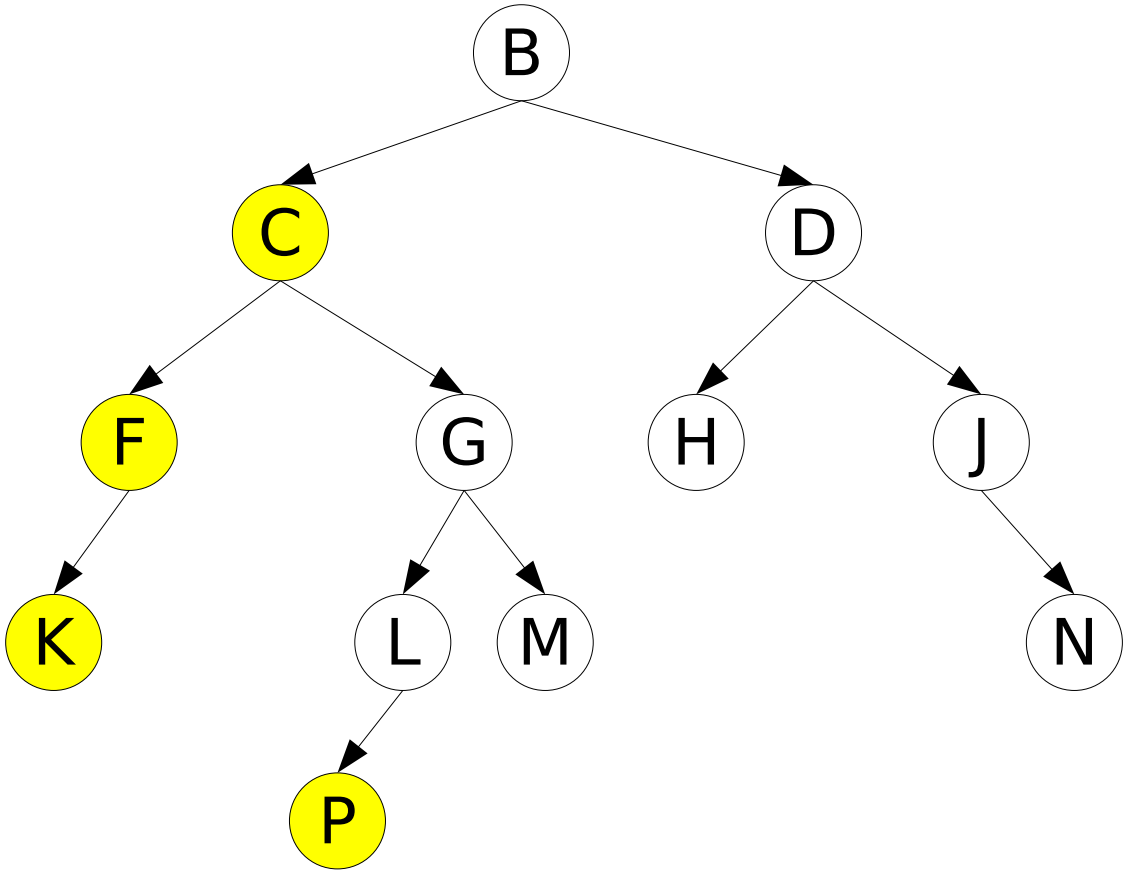
left
root
right



K F C

Inorder Traversal Example

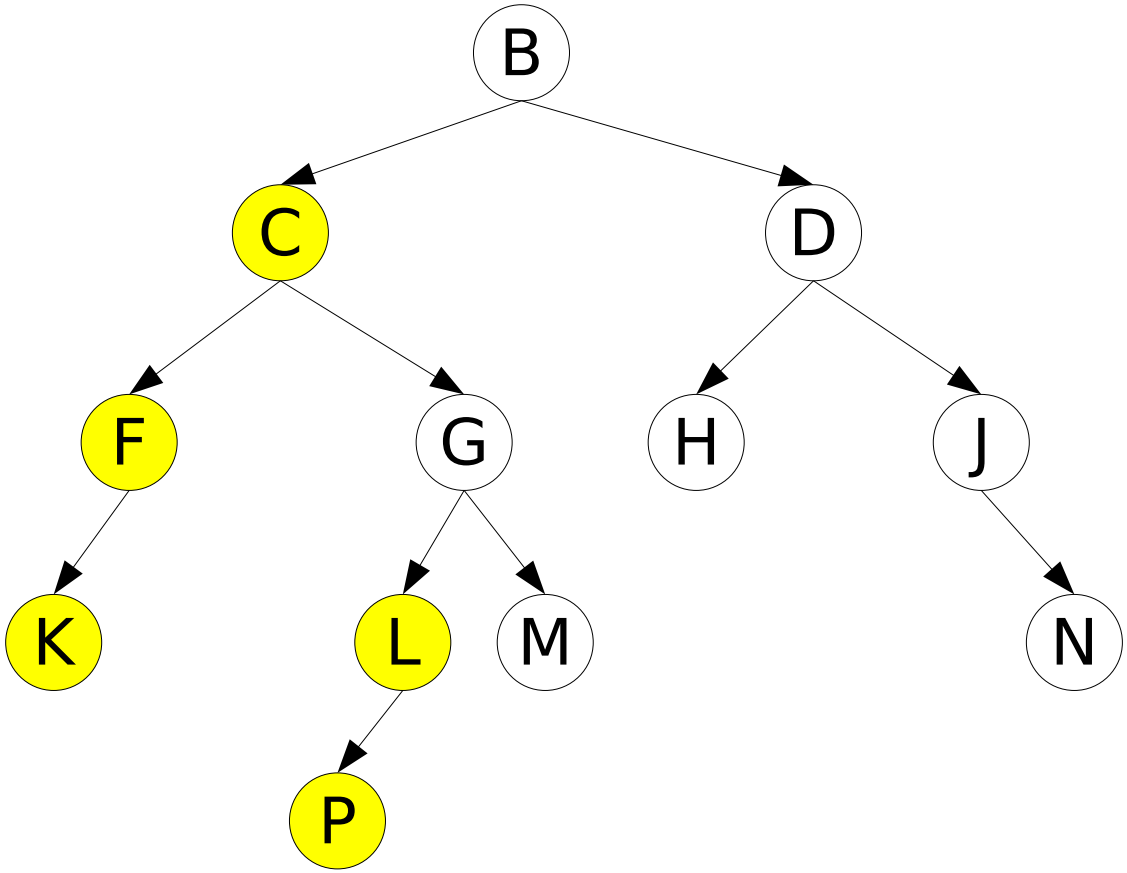
left
root
right



K F C P

Inorder Traversal Example

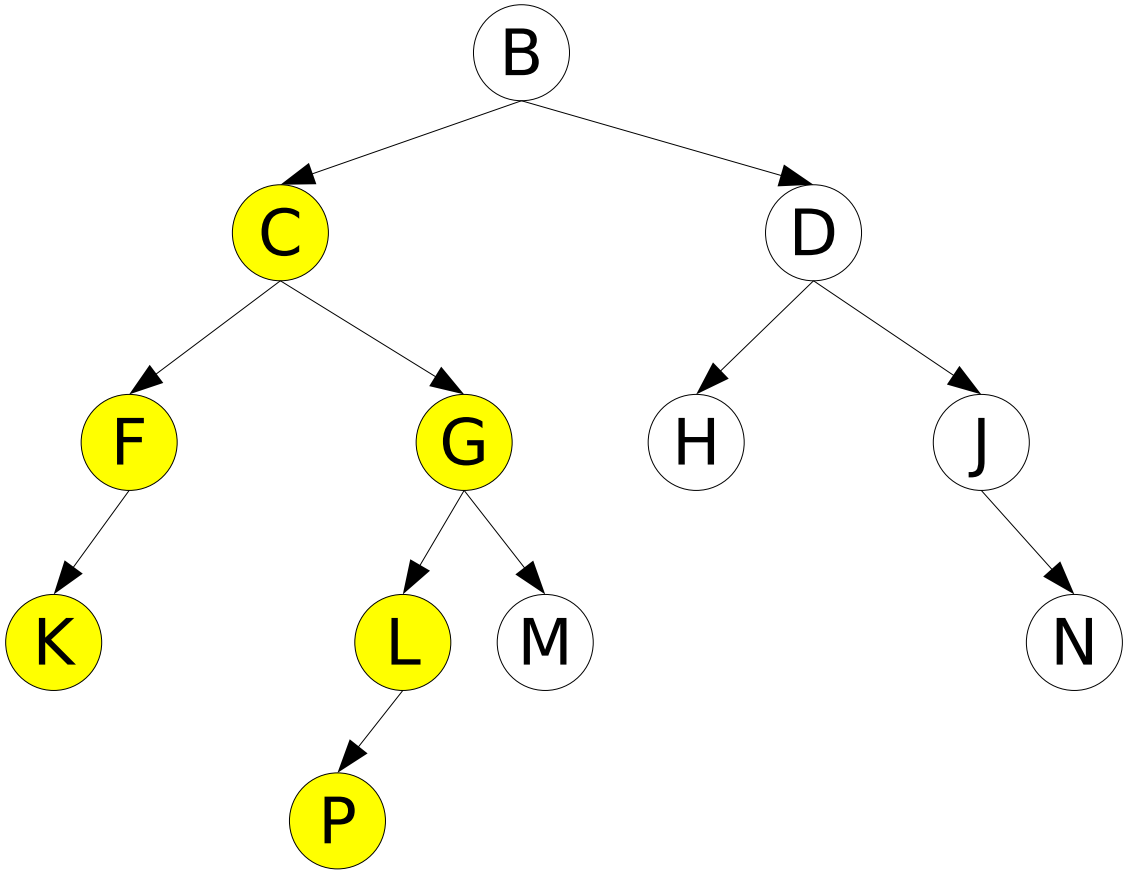
left
root
right



K F C P L

Inorder Traversal Example

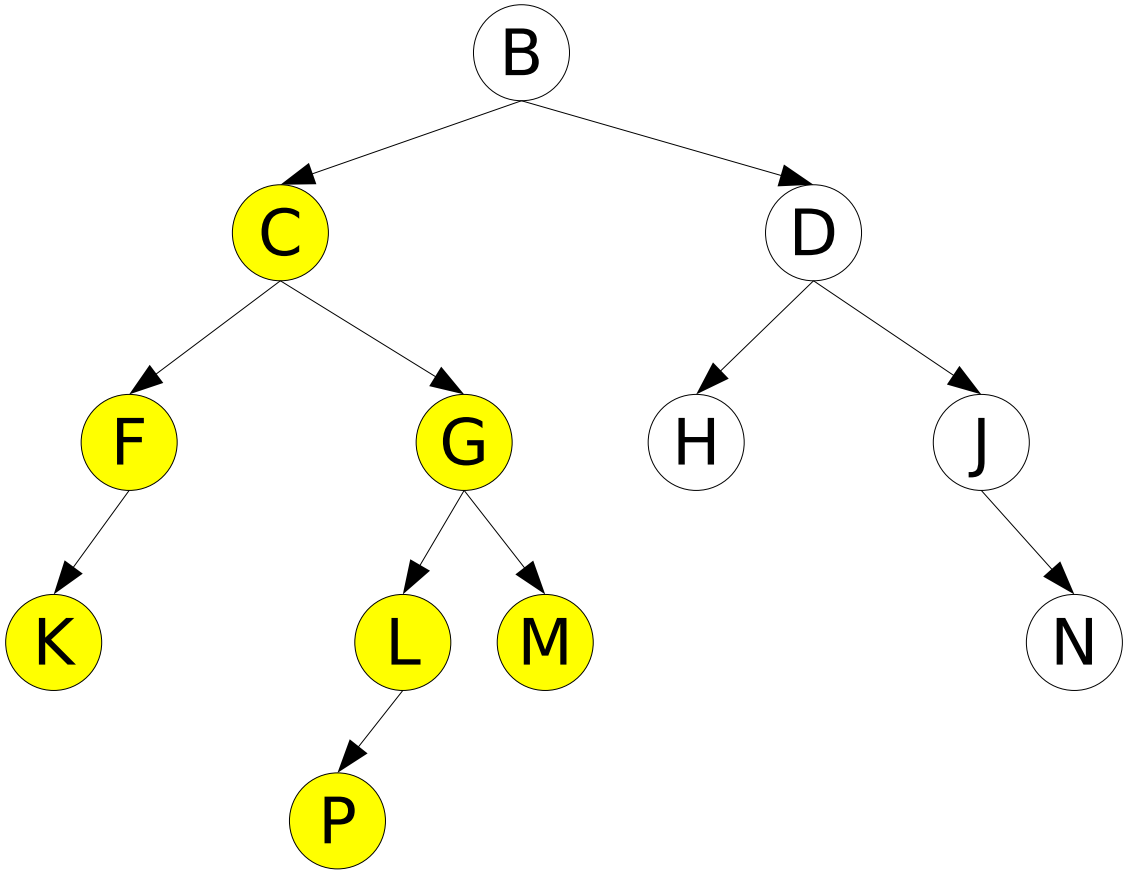
left
root
right



K F C P L G

Inorder Traversal Example

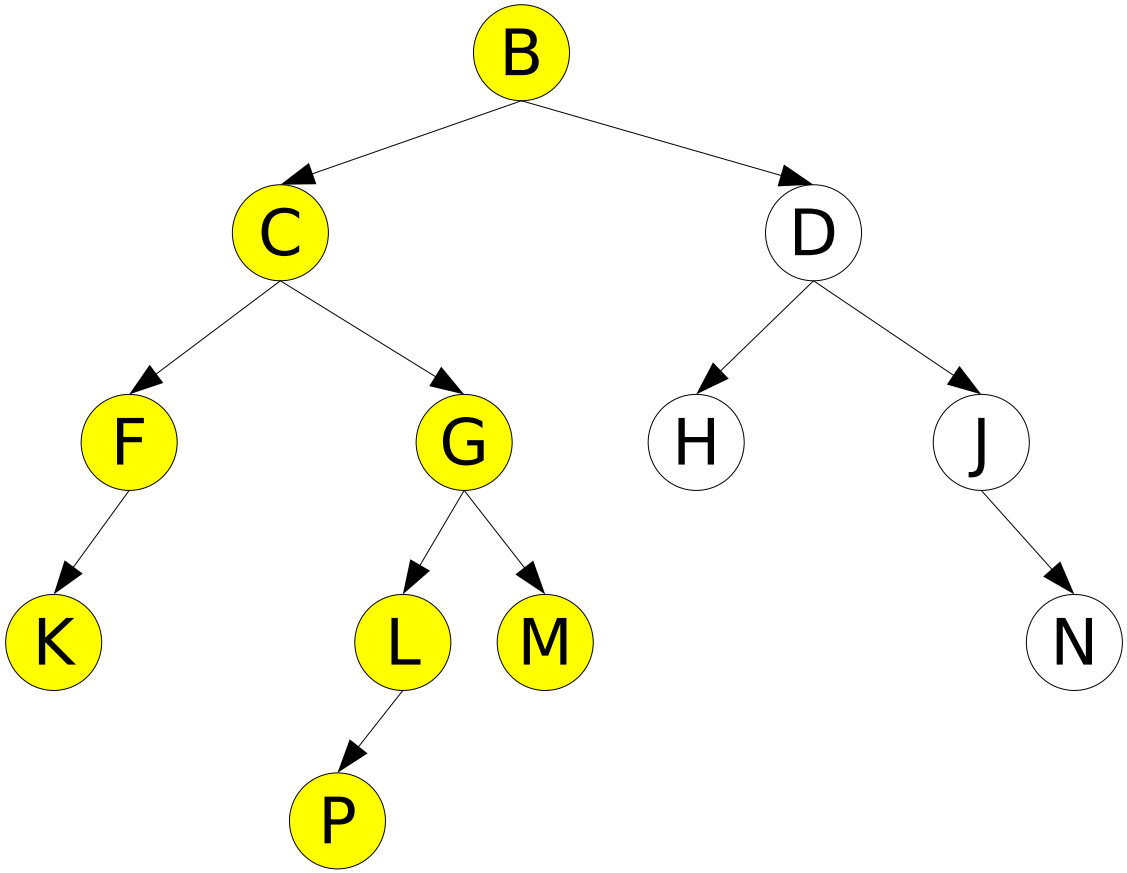
left
root
right



K F C P L G M

Inorder Traversal Example

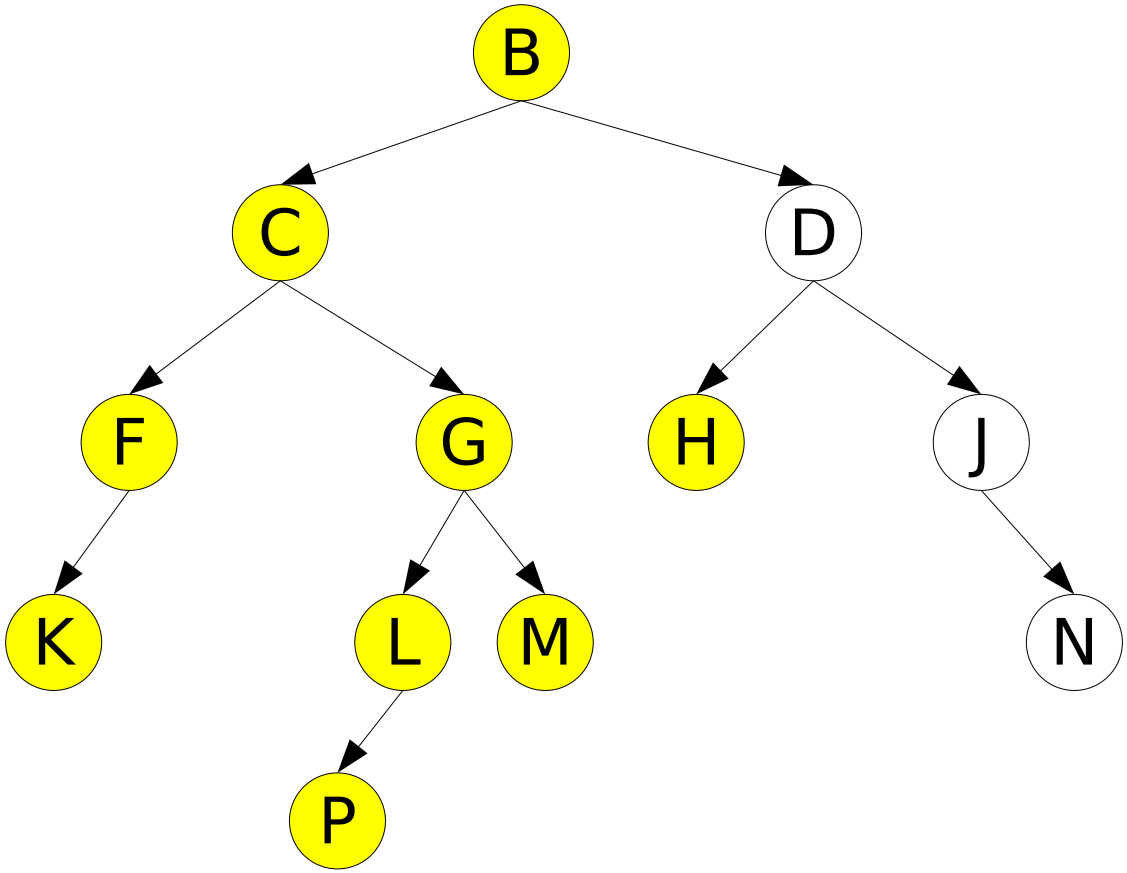
left
root
right



K F C P L G M B

Inorder Traversal Example

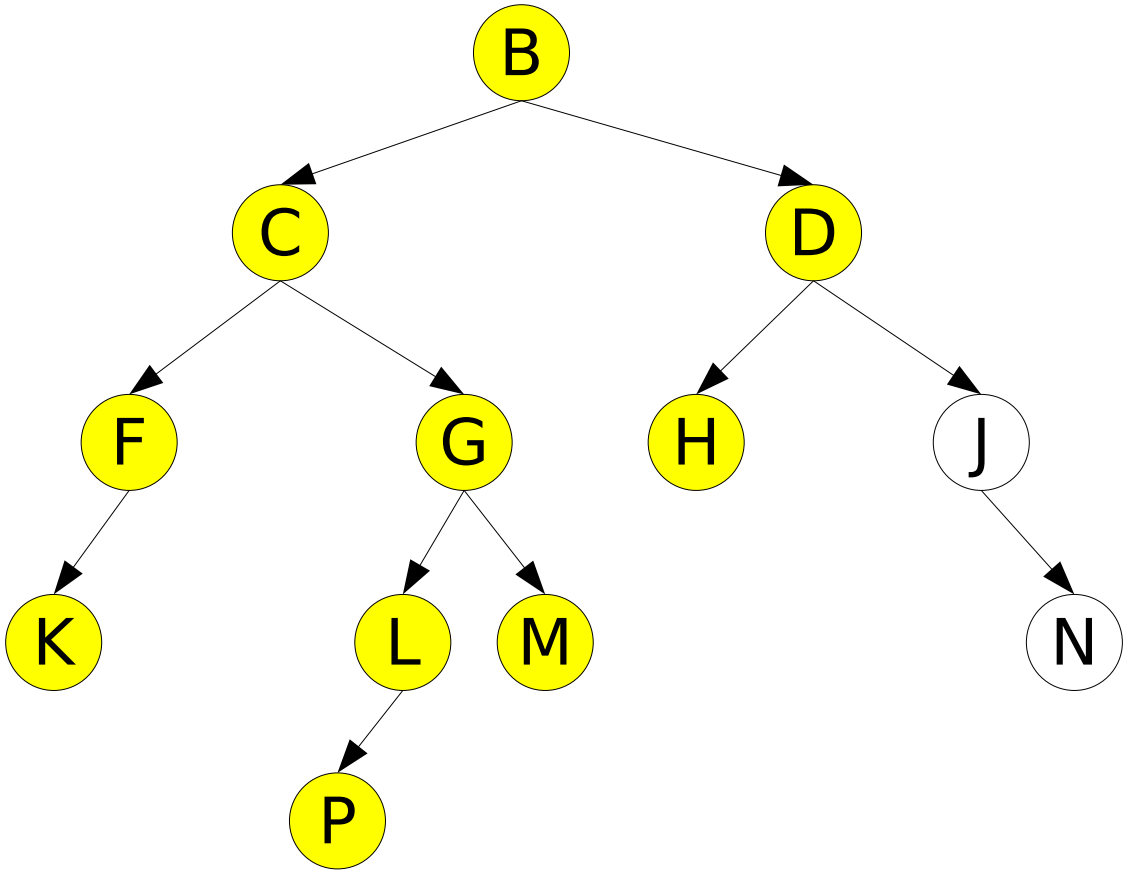
left
root
right



K F C P L G M B H

Inorder Traversal Example

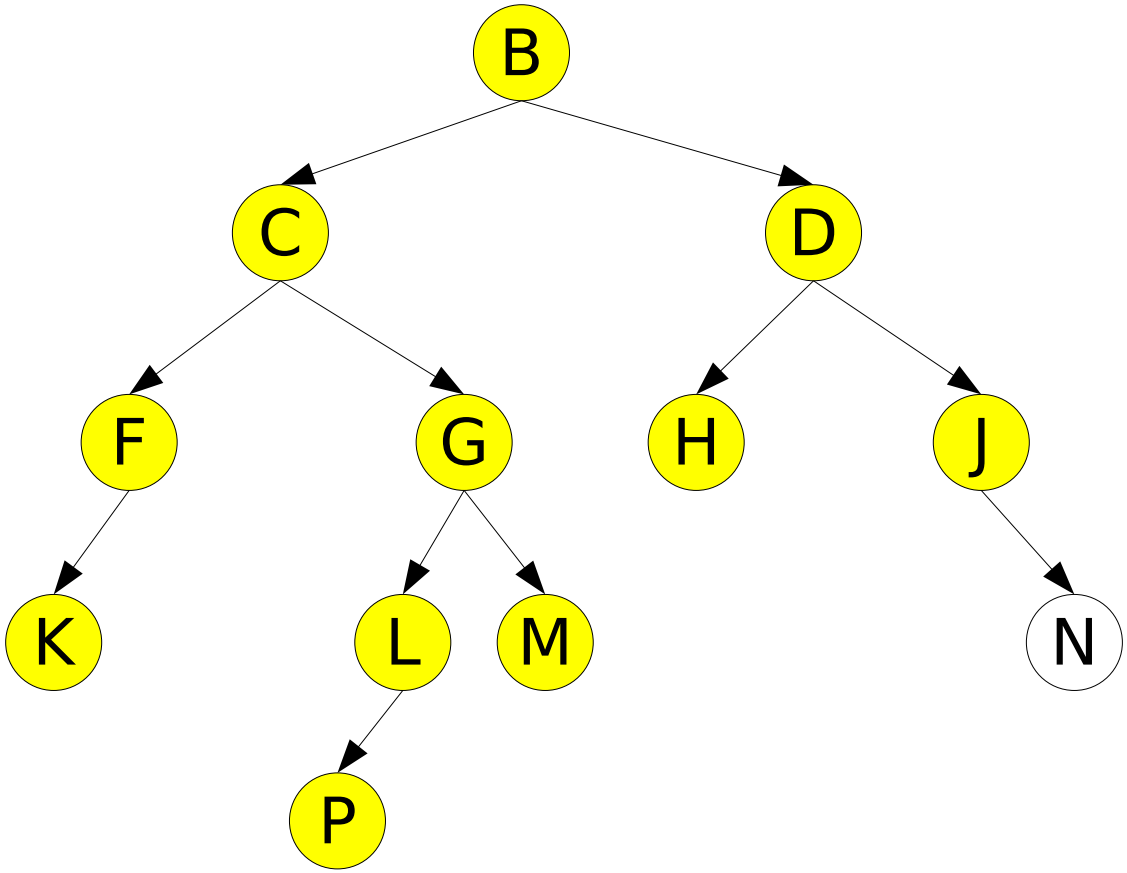
left
root
right



K F C P L G M B H D

Inorder Traversal Example

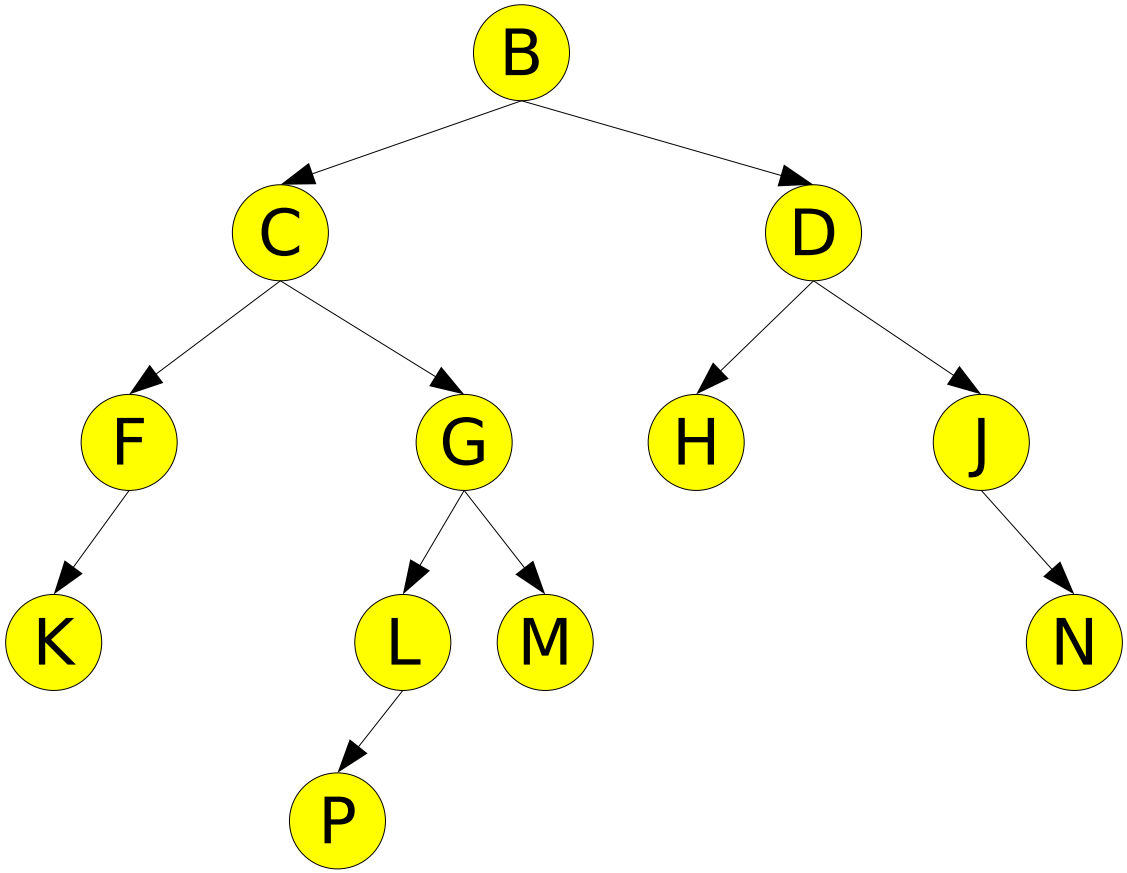
left
root
right



K F C P L G M B H D J

Inorder Traversal Example

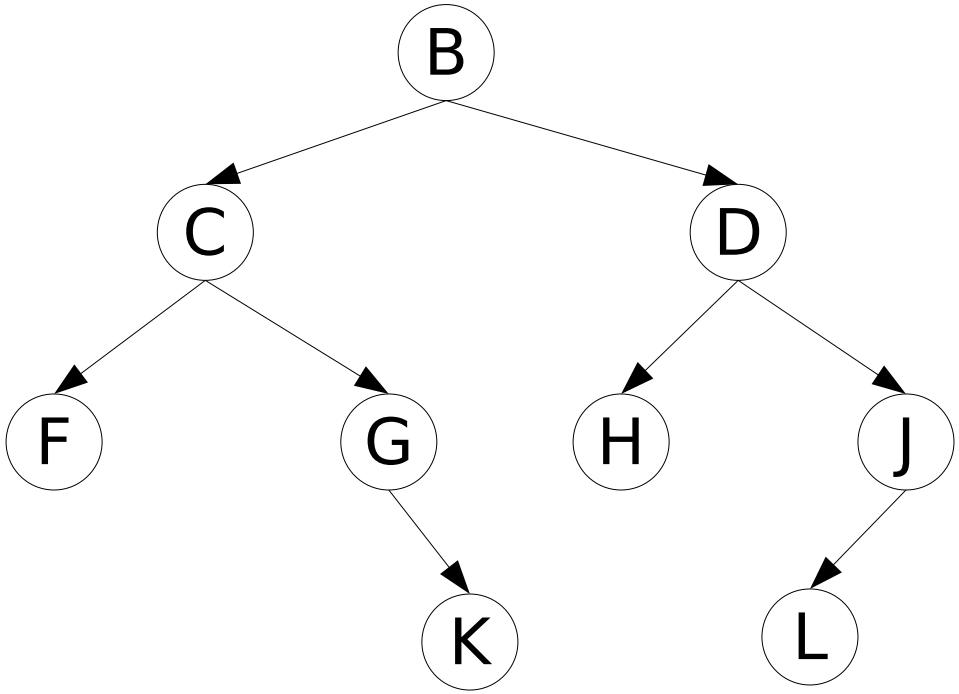
left
root
right



K F C P L G M B H D J N

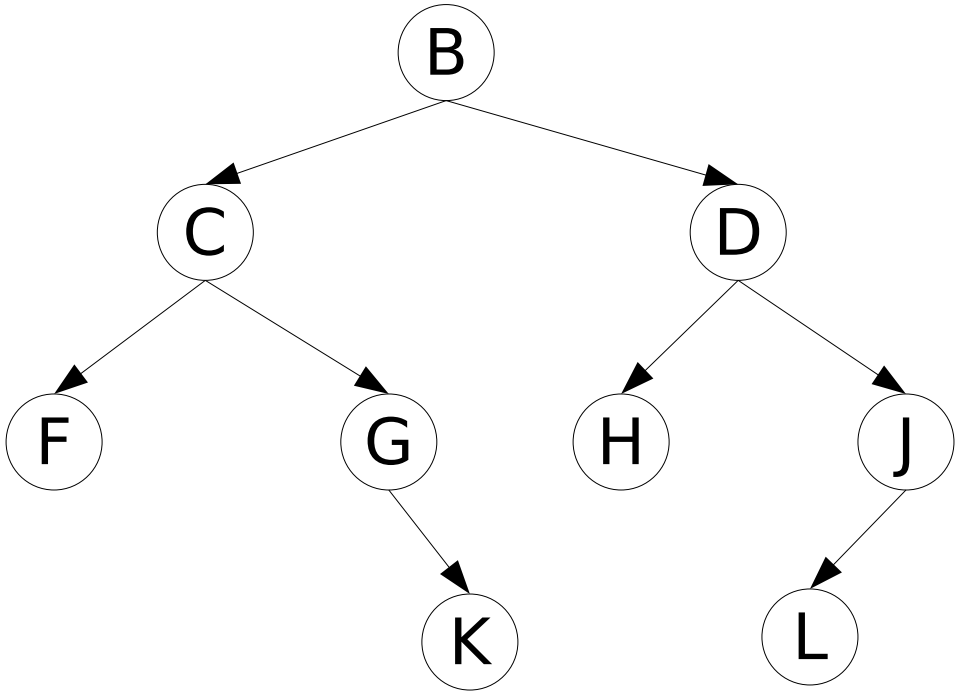
Inorder Traversal Exercise

What is the inorder traversal of this tree?



Inorder Traversal Exercise

What is the inorder traversal of this tree?



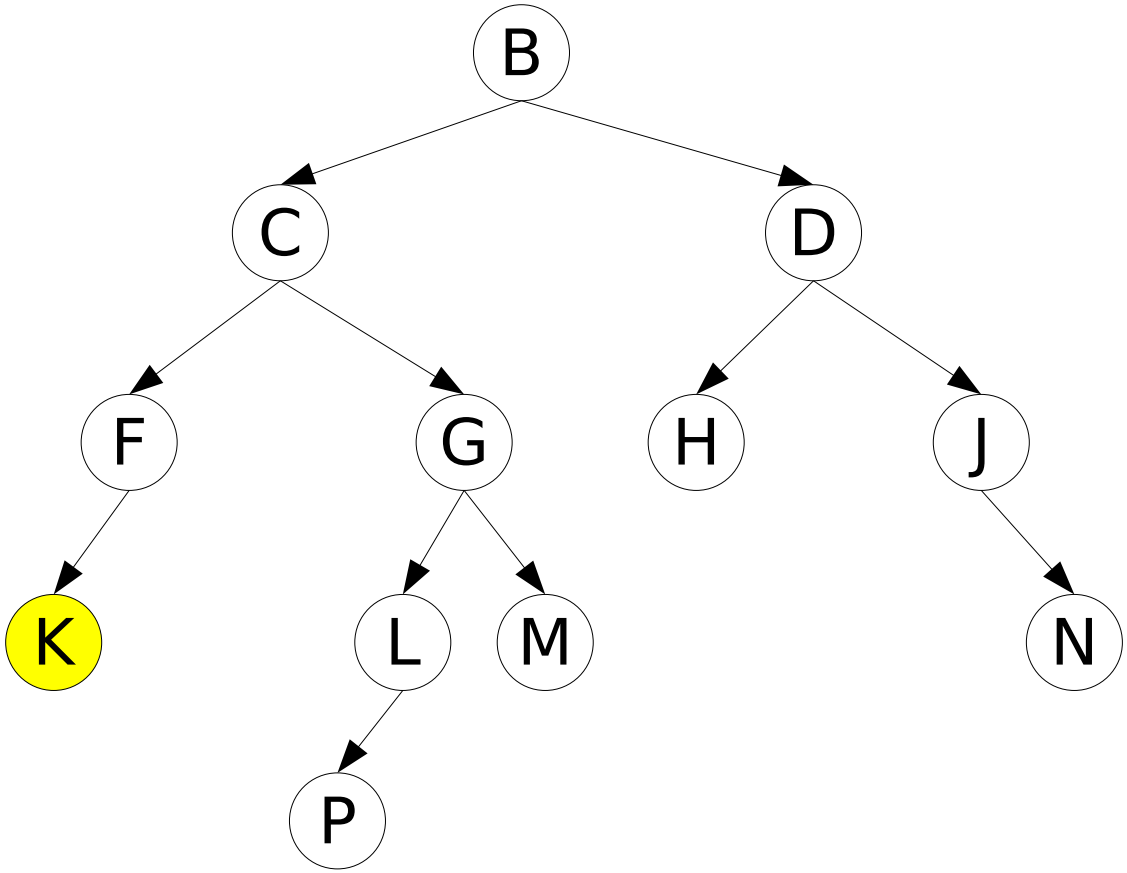
F C G K B H D L J

Postorder Traversal

- In a postorder traversal, the root node is visited after its subtrees:
  - visit all nodes in the left subtree
  - visit all nodes in the right subtree
  - **visit the root**

Postorder Traversal Example

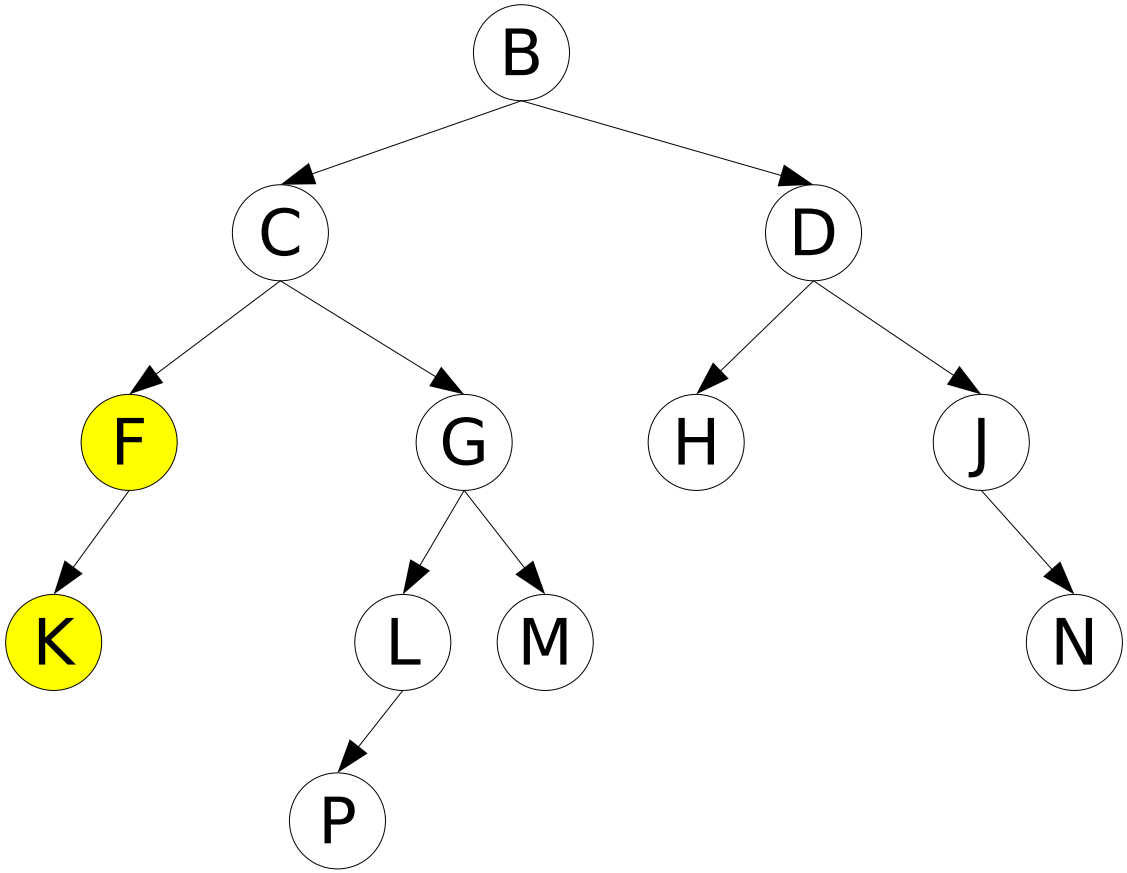
left
right
root



K

Postorder Traversal Example

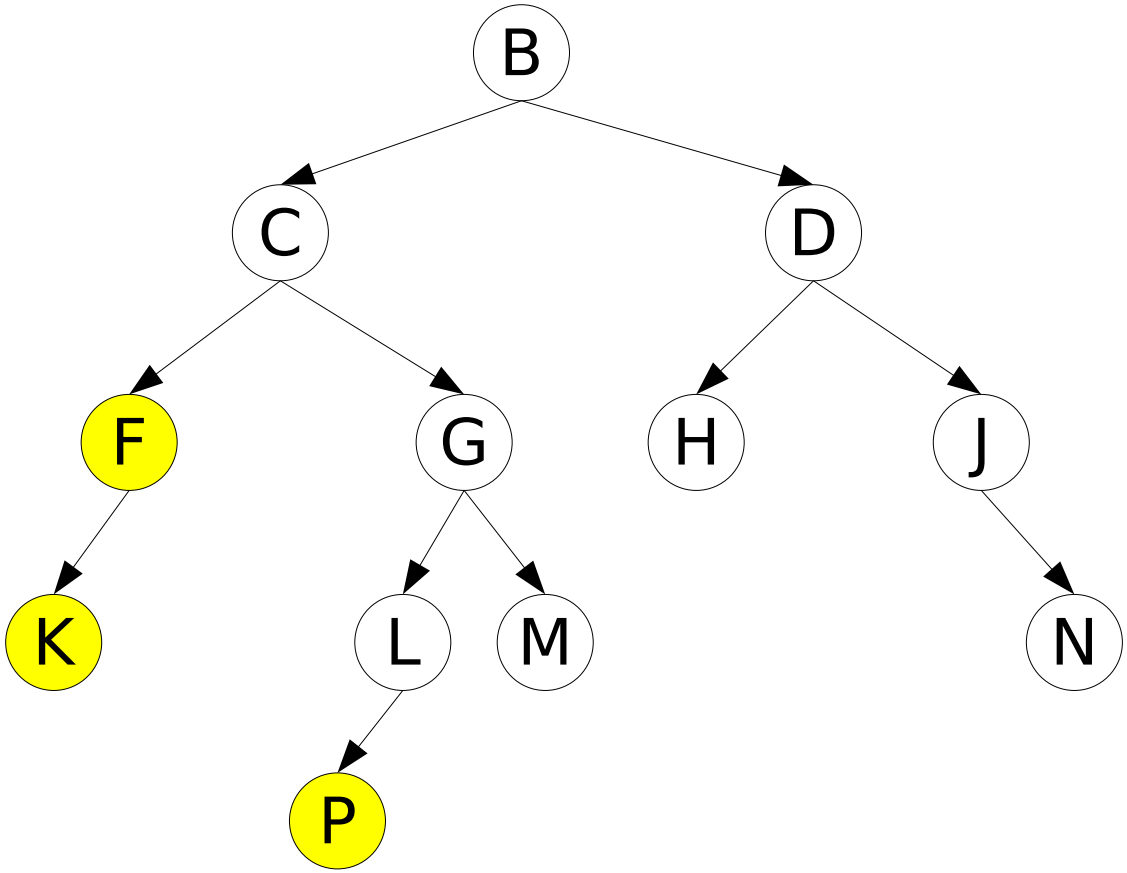
left
right
root



K F

Postorder Traversal Example

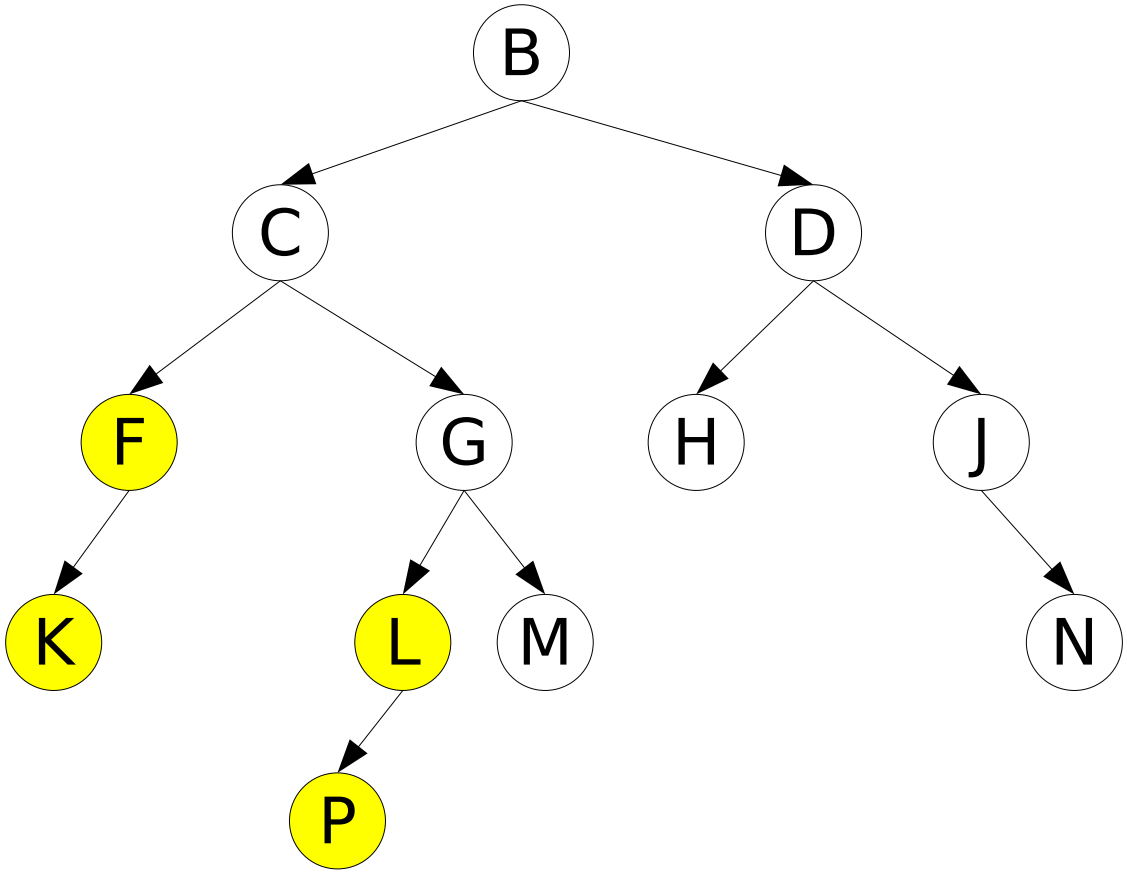
left
right
root



K F P

Postorder Traversal Example

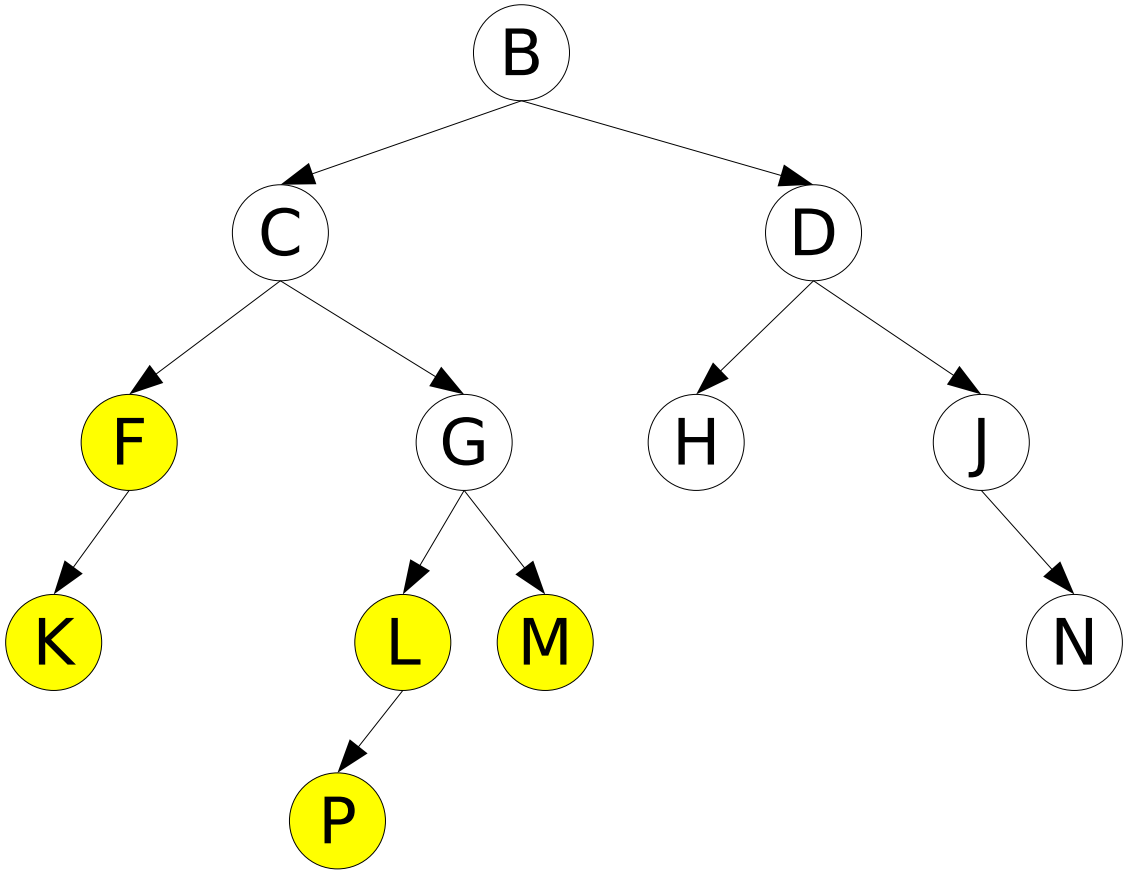
left
right
root



K F P L

Postorder Traversal Example

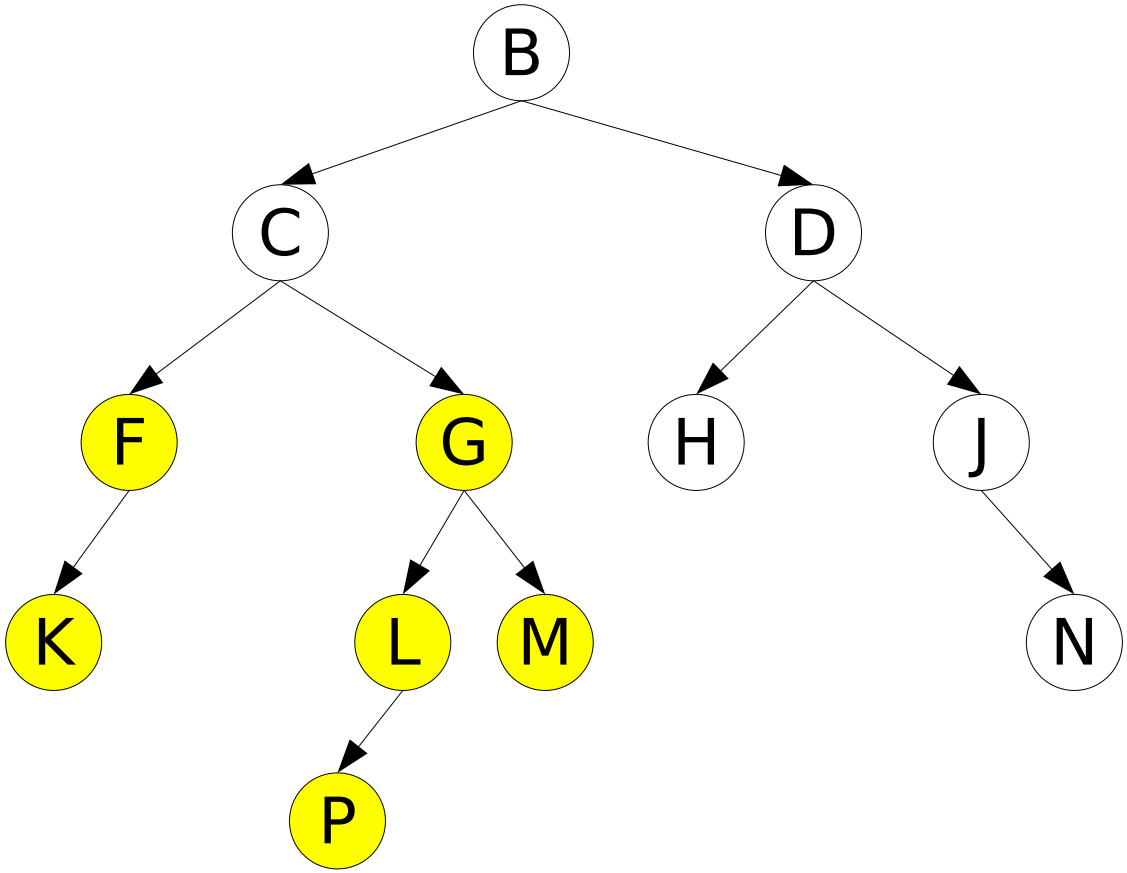
left
right
root



K F P L M

Postorder Traversal Example

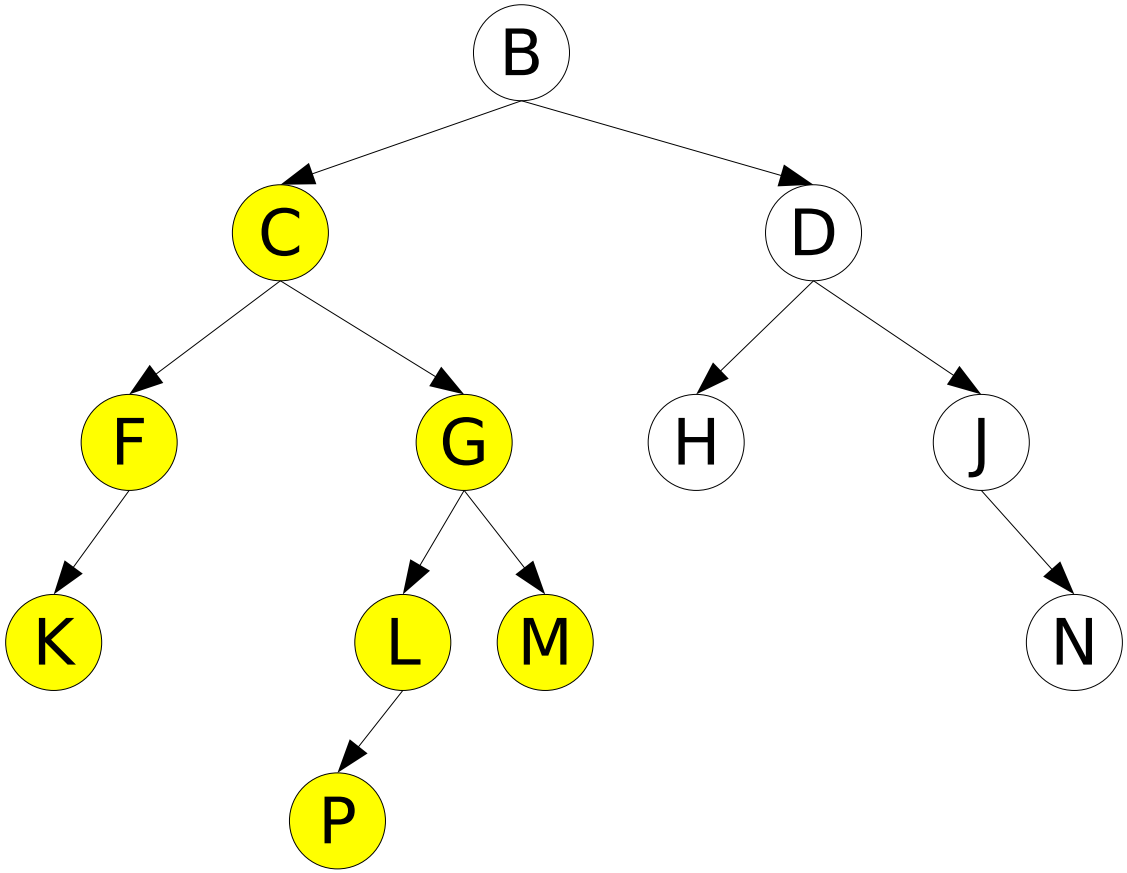
left
right
root



K F P L M G

Postorder Traversal Example

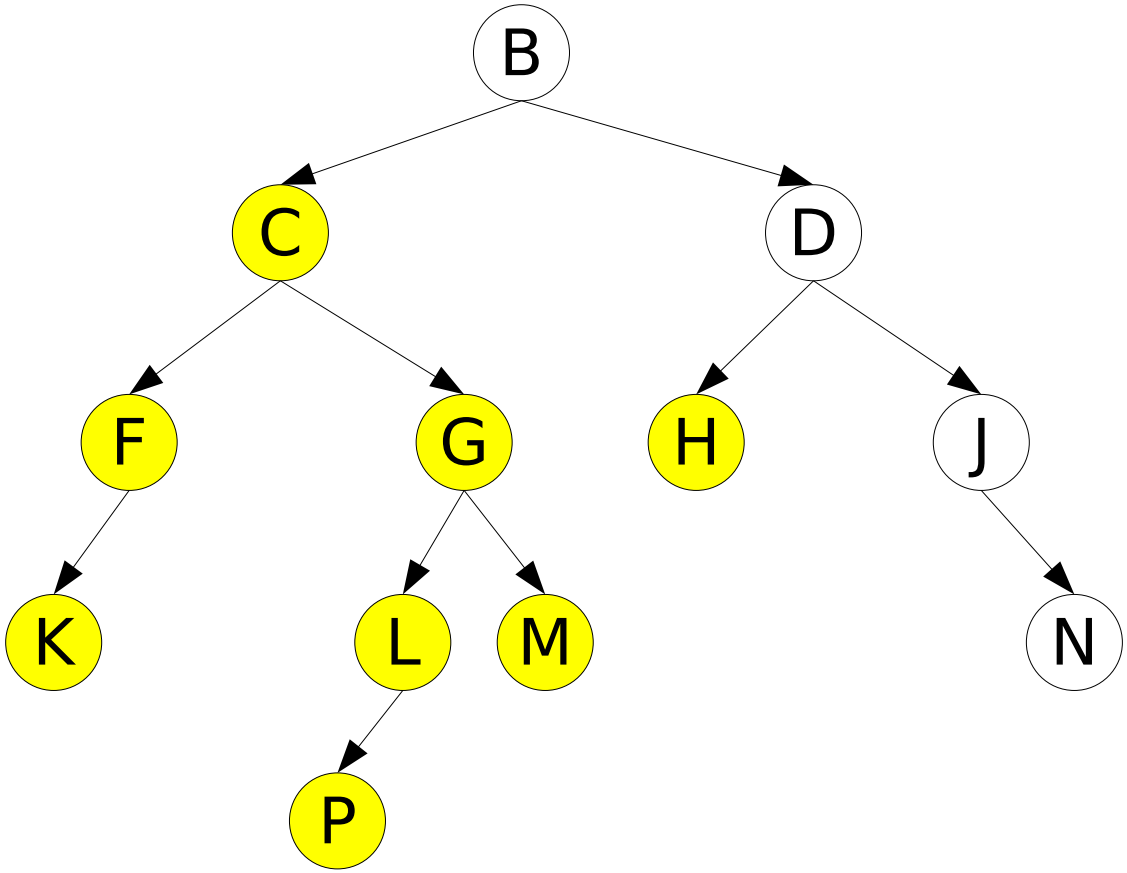
left
right
root



K F P L M G C

Postorder Traversal Example

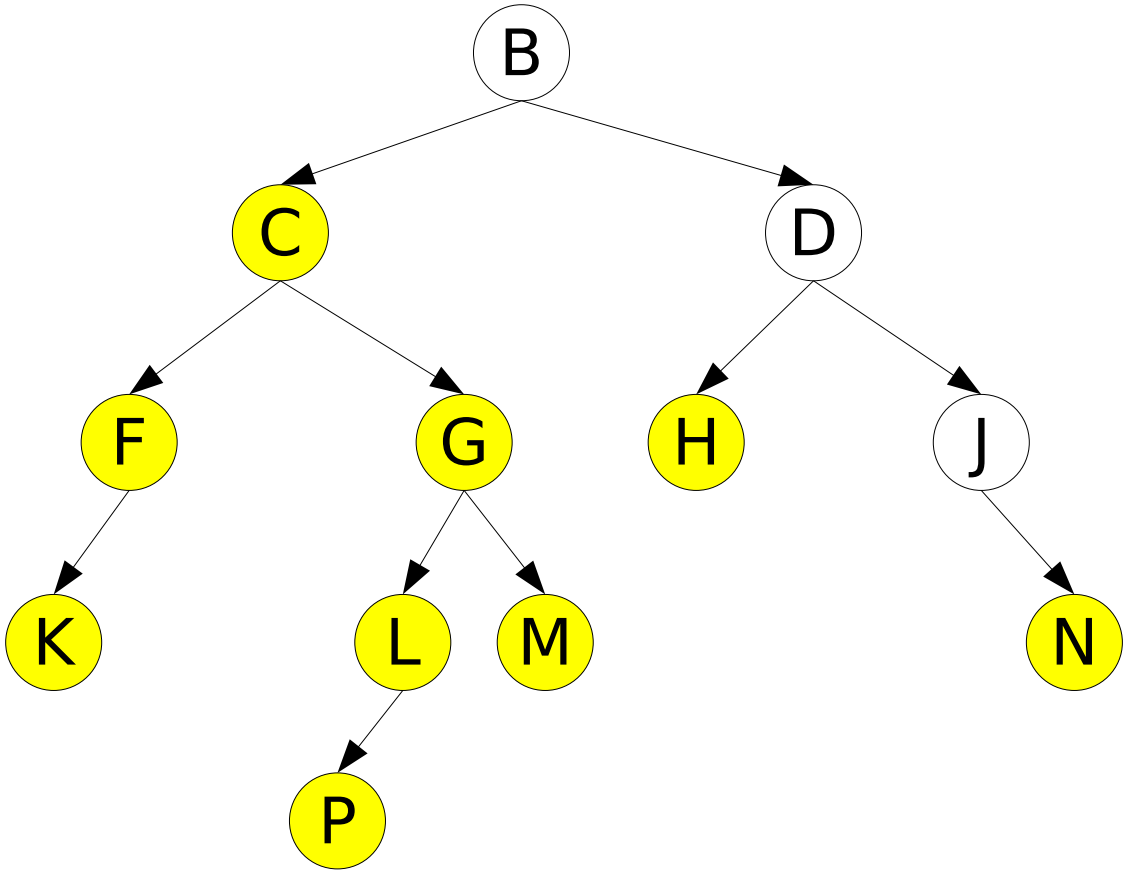
left
right
root



K F P L M G C H

Postorder Traversal Example

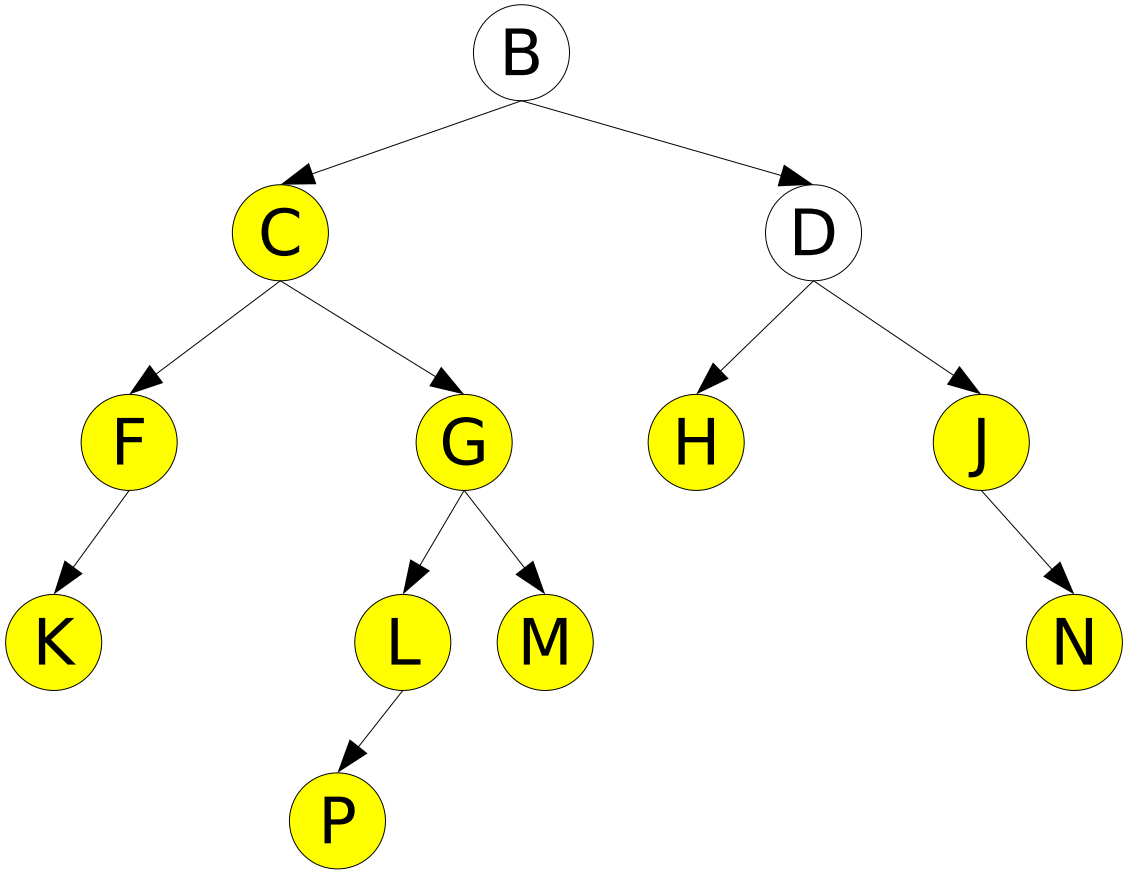
left
right
root



K F P L M G C H N

Postorder Traversal Example

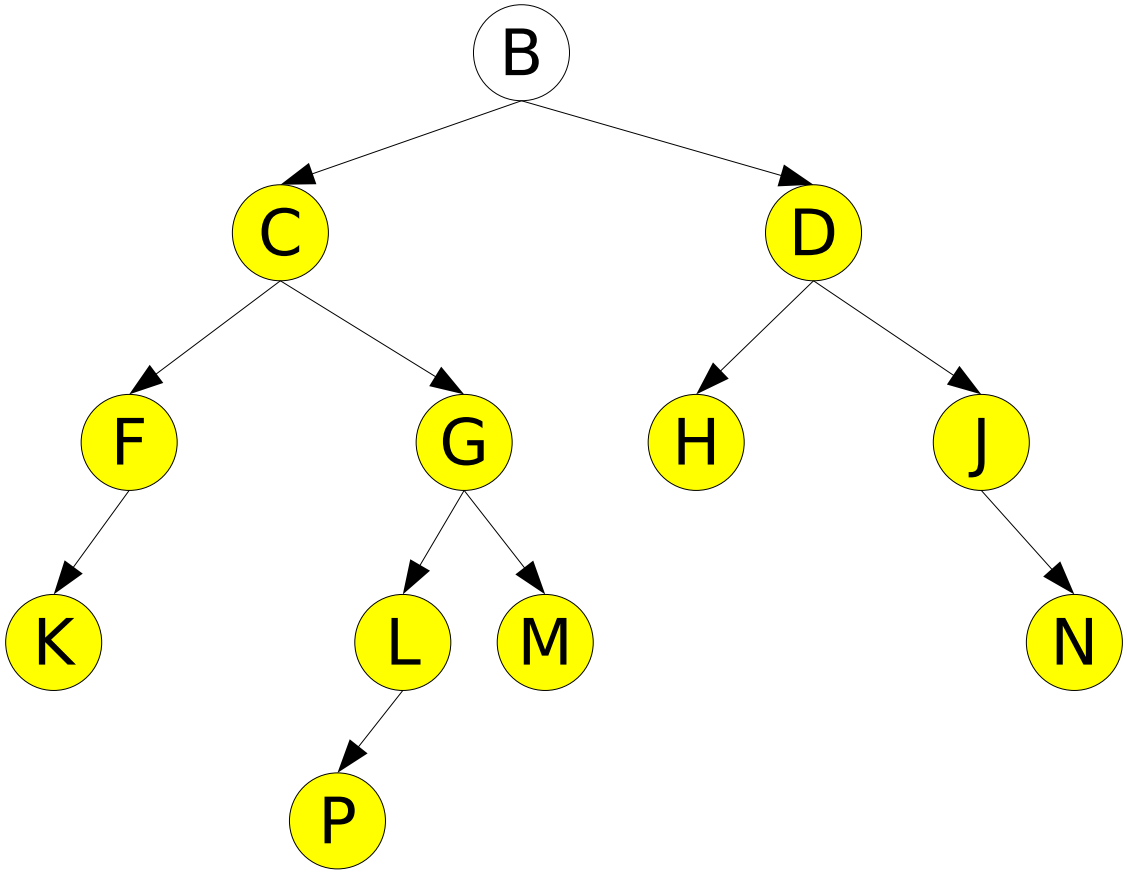
left
right
root



K F P L M G C H N J

Postorder Traversal Example

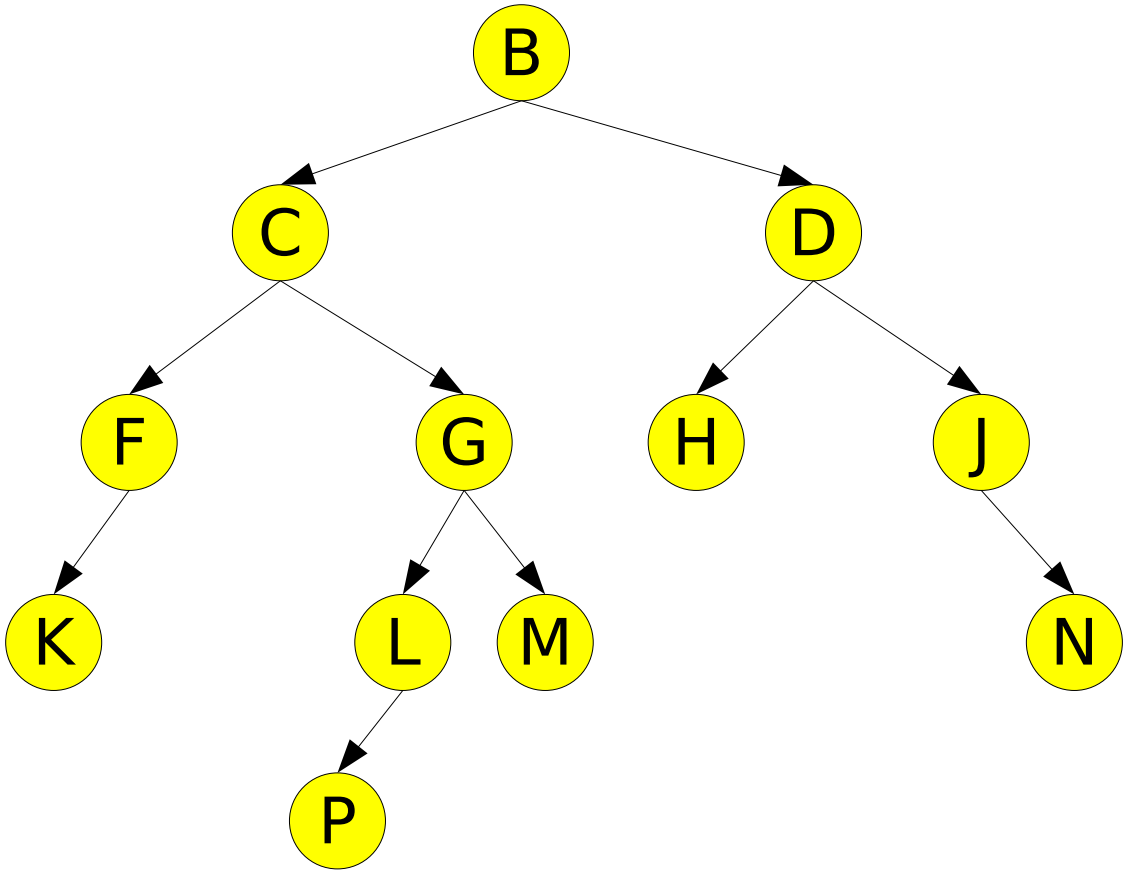
left
right
root



K F P L M G C H N J D

Postorder Traversal Example

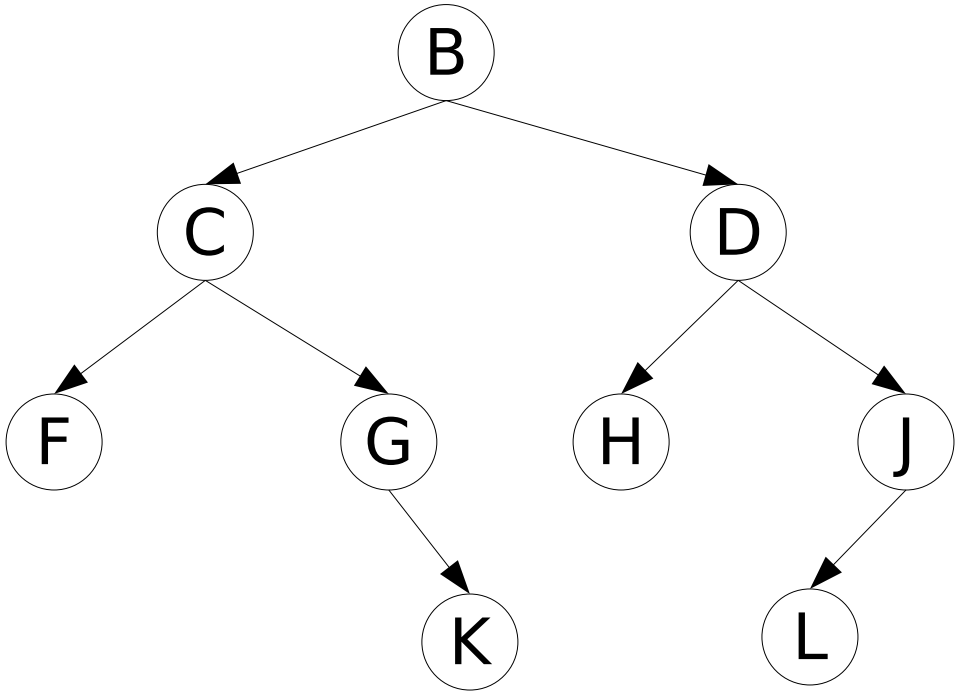
left
right
root



K F P L M G C H N J D B

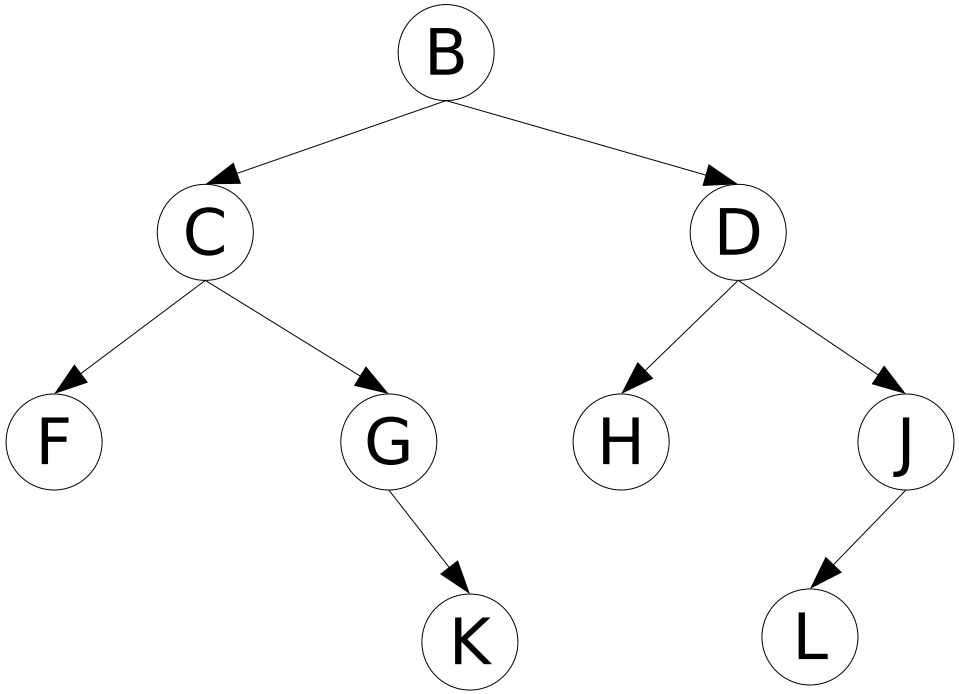
Postorder Traversal Exercise

What is the postorder traversal of this tree?



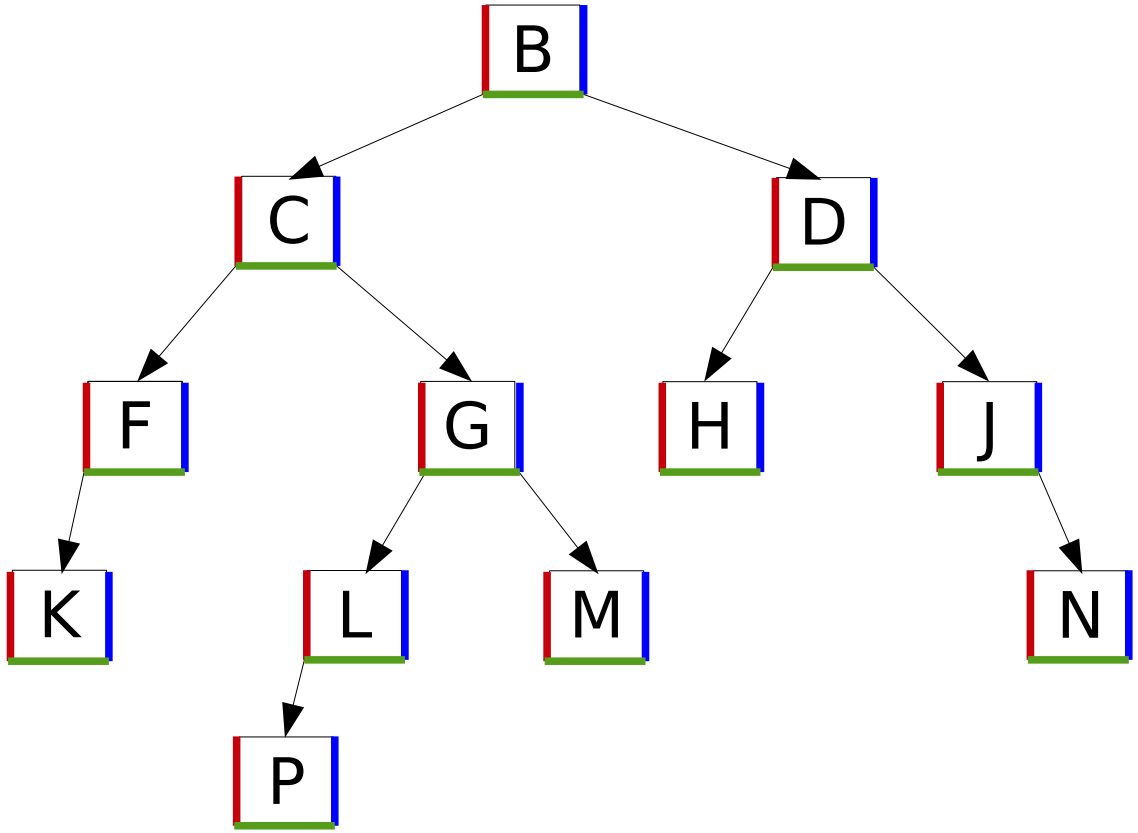
Postorder Traversal Exercise

What is the postorder traversal of this tree?

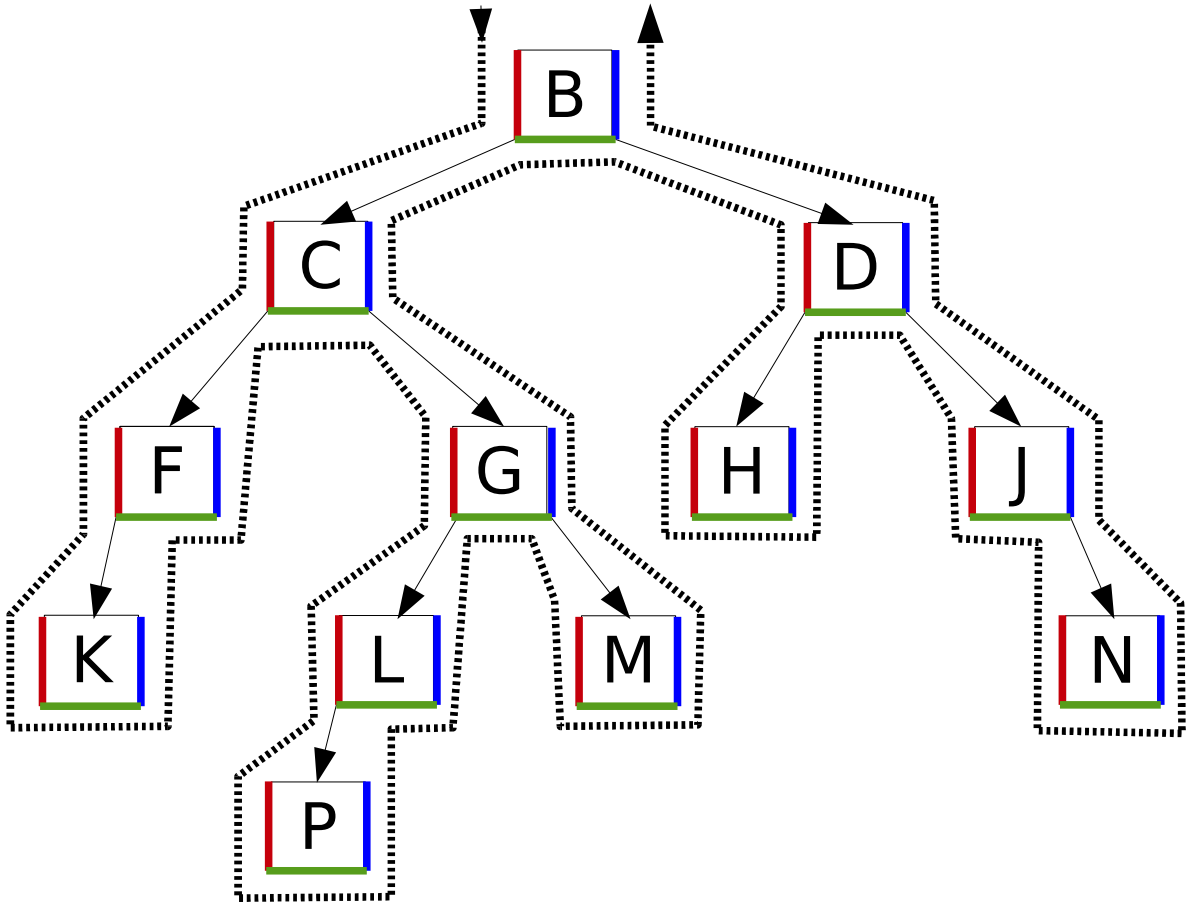


F K G C H L J D B

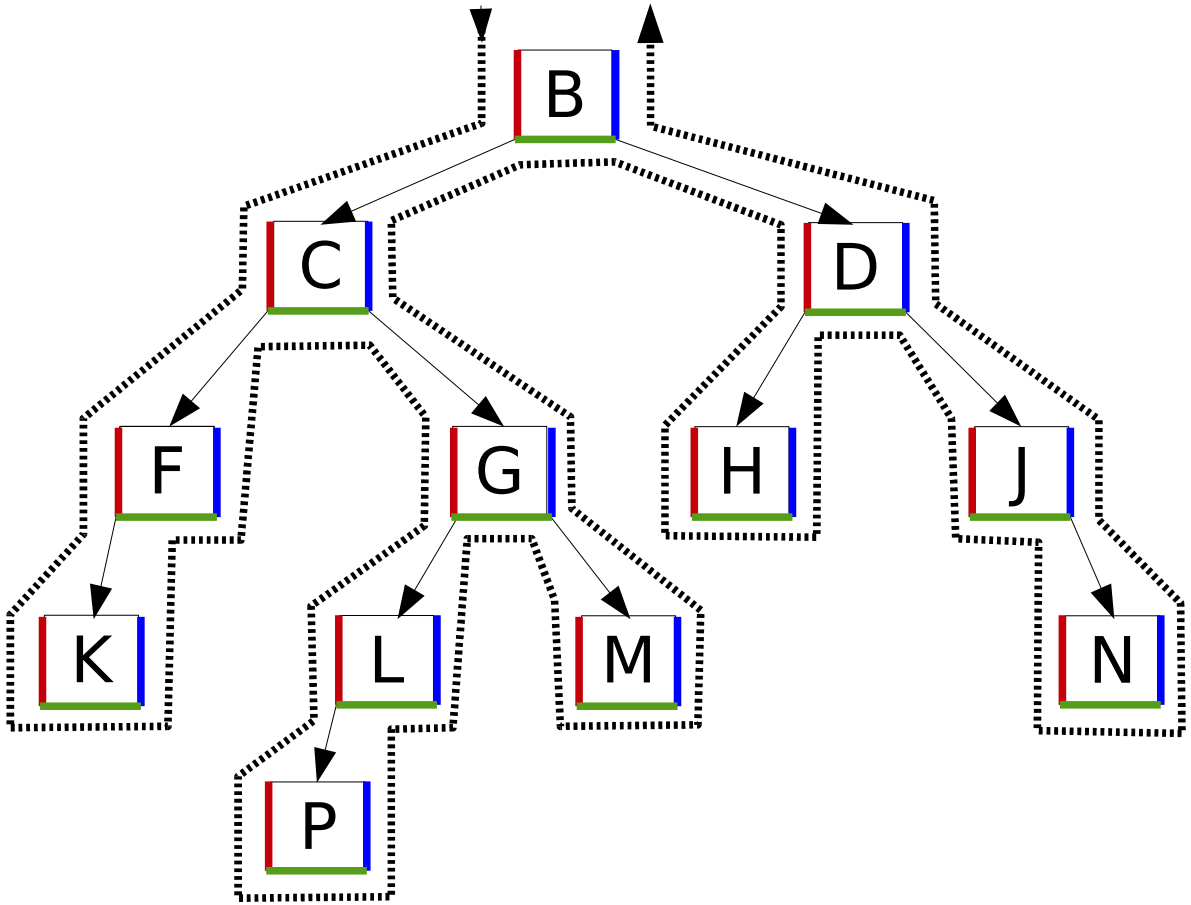
Distinguishing the Traversal Types



Distinguishing the Traversal Types



Distinguishing the Traversal Types



Preorder (left/red window): B C F K G L P M D H J N

Inorder (bottom/green window): K F C P L G M B H D J N

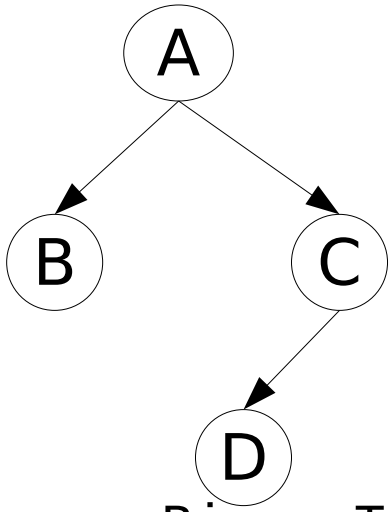
Postorder (right/blue window): K F P L M G C H N J D B

Building a Binary Tree

- Unlike classes that represent lists, stacks, or queues, tree classes often do **not** have methods to add or remove elements.
- There is no obvious place to add a new element.
- Removing a node is even less clear
  - how would you indicate which node should be removed? - can't number them like in a list
  - what happens to the children of the removed node?

Building a Binary Tree

- Trees are built up, node by node.
- Let's build this tree:



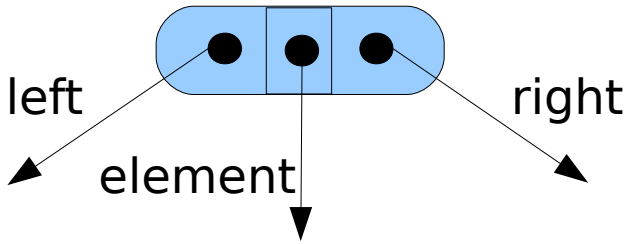

```
BinaryTree<String> b = new BinaryTree<String>("B");
BinaryTree<String> d = new BinaryTree<String>("D");
BinaryTree<String> c = new BinaryTree<String>("C", d, null);
BinaryTree<String> a = new BinaryTree<String>("A", b, c);
```

BinaryTreeNode<T> Instance Variables

```
public class BinaryTreeNode<T> {
    T element;
    BinaryTreeNode<T> left;
    BinaryTreeNode<T> right;

    ...

}
```



BinaryTreeNode<T> Constructors

```
public class BinaryTreeNode<T> {
    T element;
    BinaryTreeNode<T> left;
    BinaryTreeNode<T> right;

    public BinaryTreeNode(T dataObj) {
        element = dataObj;
        left = right = null;
    }

    public BinaryTreeNode(T dataObj,
                          BinaryTreeNode<T> l,
                          BinaryTreeNode<T> r) {
        element = dataObj;
        left = l;
        right = r;
    }
}
```

BinaryTreeNode<T> toString

```
public class BinaryTreeNode<T> {
    T element;
    BinaryTreeNode<T> left;
    BinaryTreeNode<T> right;

    <Constructors>

    public String toString() {
        return element.toString();
    }
}
```

BinaryTreeNode<T> isLeaf - Exercise

```
public class BinaryTreeNode<T> {
    T element;
    BinaryTreeNode<T> left;
    BinaryTreeNode<T> right;

    <Constructors>

    public boolean isLeaf() {

        //************  To Do *********

    }
}
```

BinaryTreeNode<T> isLeaf

```
public class BinaryTreeNode<T> {
    T element;
    BinaryTreeNode<T> left;
    BinaryTreeNode<T> right;

    <Constructors>

    public boolean isLeaf() {

        return (left == null && right == null);

    }
}
```

BinaryTree<T>

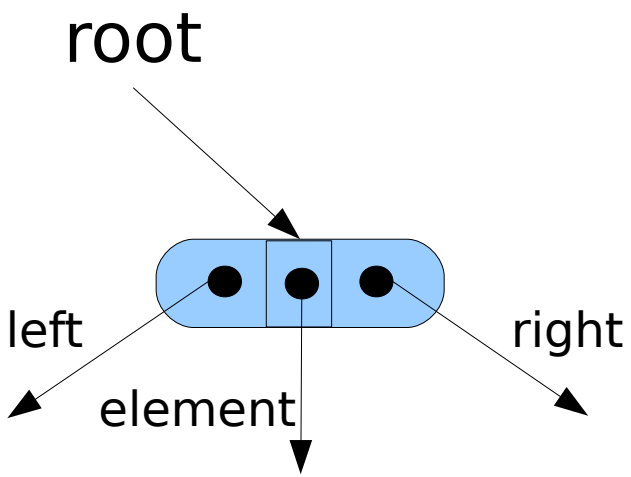
- Now that we have a class to represent a node in the tree, we can define a class to represent the tree itself.
- The only node that we need a reference to is the root node. All other nodes are accessible from the root.
- We will declare a reference to the root node as an instance variable.

BinaryTree<T> Instance Variable

```
public class BinaryTree<T> {
    BinaryTreeNode<T> root;

    ...

}
```



BinaryTree<T> Constructors

```
public class BinaryTree<T> {
    BinaryTreeNode<T> root;

    public BinaryTree() {
        root = null;
    }

    public BinaryTree(T element) {
        root = new BinaryTreeNode<T> (element);
    }

    ...
}
```

BinaryTree<T> Constructors, cont.

```
public class BinaryTree<T> {
    ...

    public BinaryTree(T element, BinaryTree<T> leftSubtree,
                                 BinaryTree<T> rightSubtree) {
        root = new BinaryTreeNode<T> (element);

        if (leftSubtree != null) {
            root.left = leftSubtree.root;
        } else {
            root.left = null;
        }

        if (rightSubtree != null) {
            root.right = rightSubtree.root;
        } else {
            root.right = null;
        }
    }
}
```