Data Structures

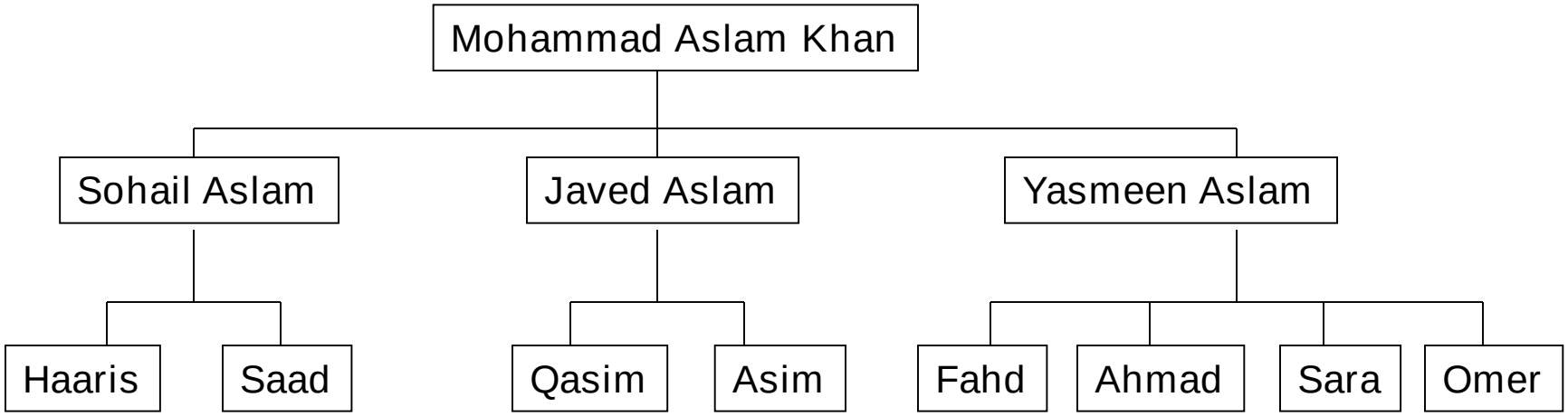
Tree

Content

- Tree Data Structure and its applications
- Terminologies of a Tree
- Binary Tree
- Complete and Full Binary Tree
- Balanced tree
- **Traversing a Tree**
  - Inorder / preorder / postorder

Tree Data Structures

- There are a number of applications where linear data structures are not appropriate.
- Consider a genealogy tree of a family.

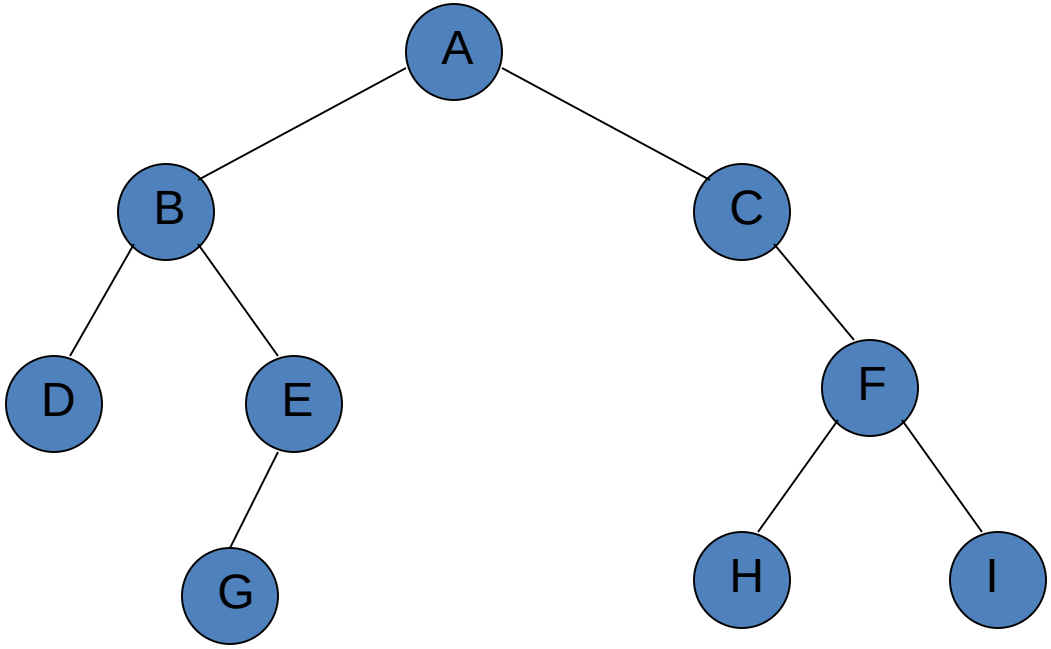


Common use of Tree data structure

- Representing **hierarchical data**
- Storing data in a way that makes it **easily searchable**
- Representing **sorted lists** of data
- As a workflow for **compositing digital images** for visual effects
- **Routing** algorithms

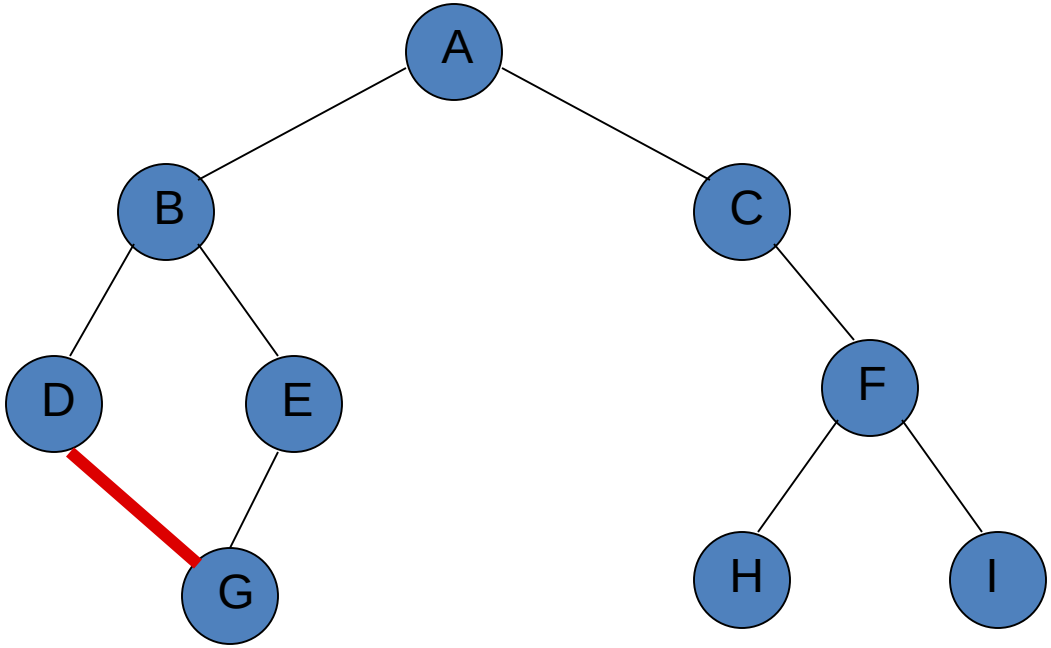
Definition of Tree Data Structure:

As a data structure, a tree is a group of nodes, where each node has a value and a list of references to other nodes (its children nodes).



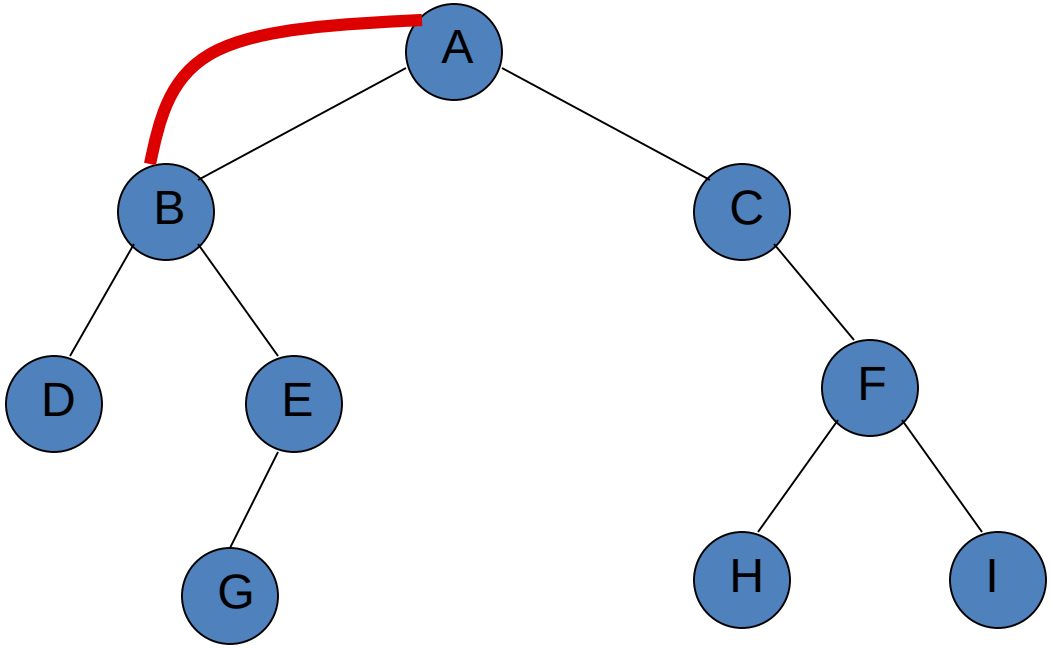
Not a Tree

- Structures that are not trees.



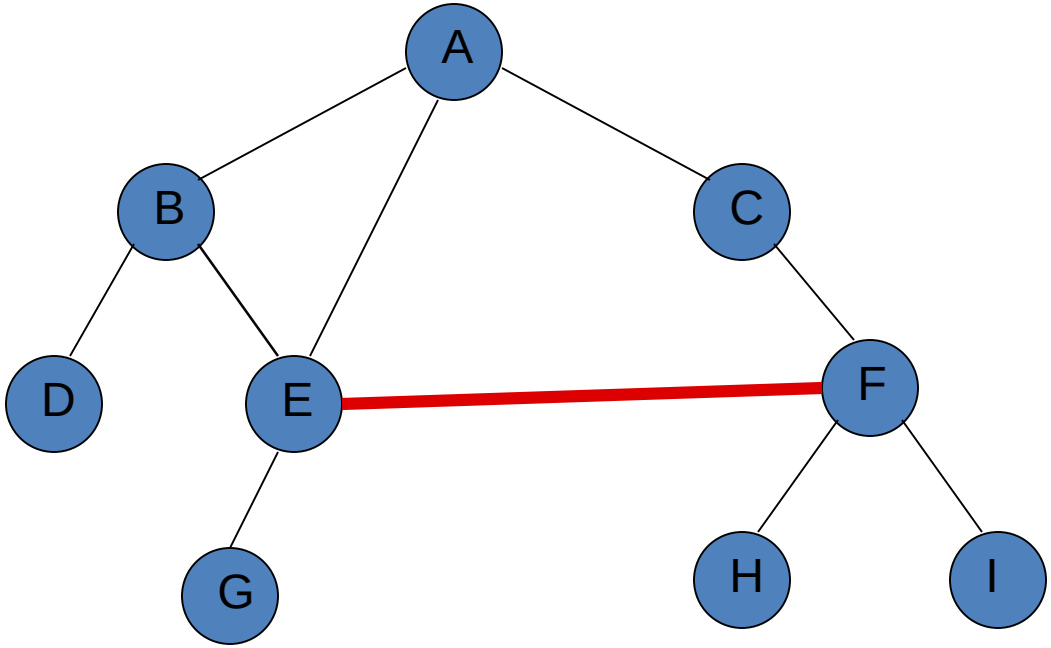
Not a Tree

- Structures that are not trees.

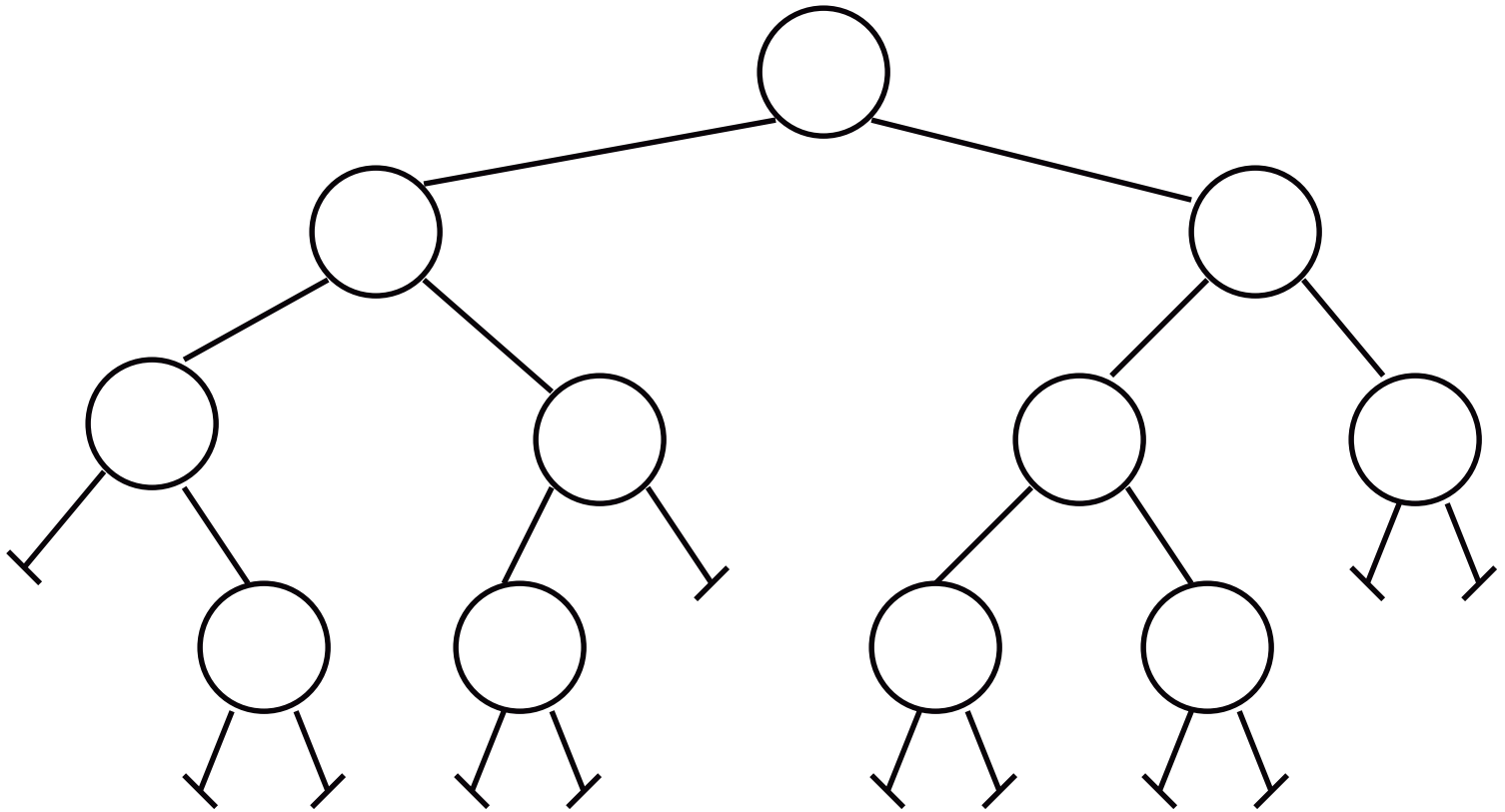


Not a Tree

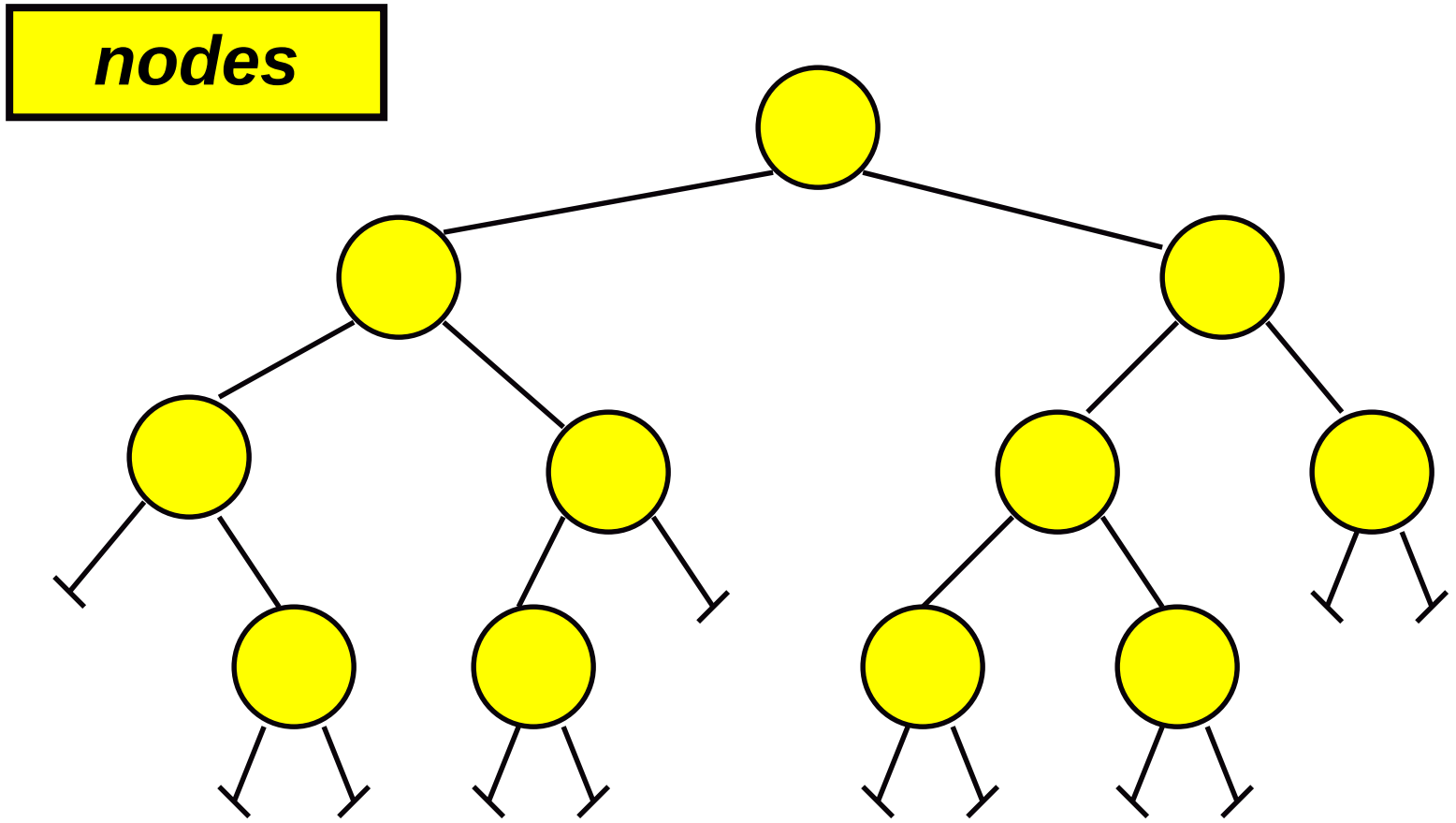
- Structures that are not trees.



Parts of a Tree

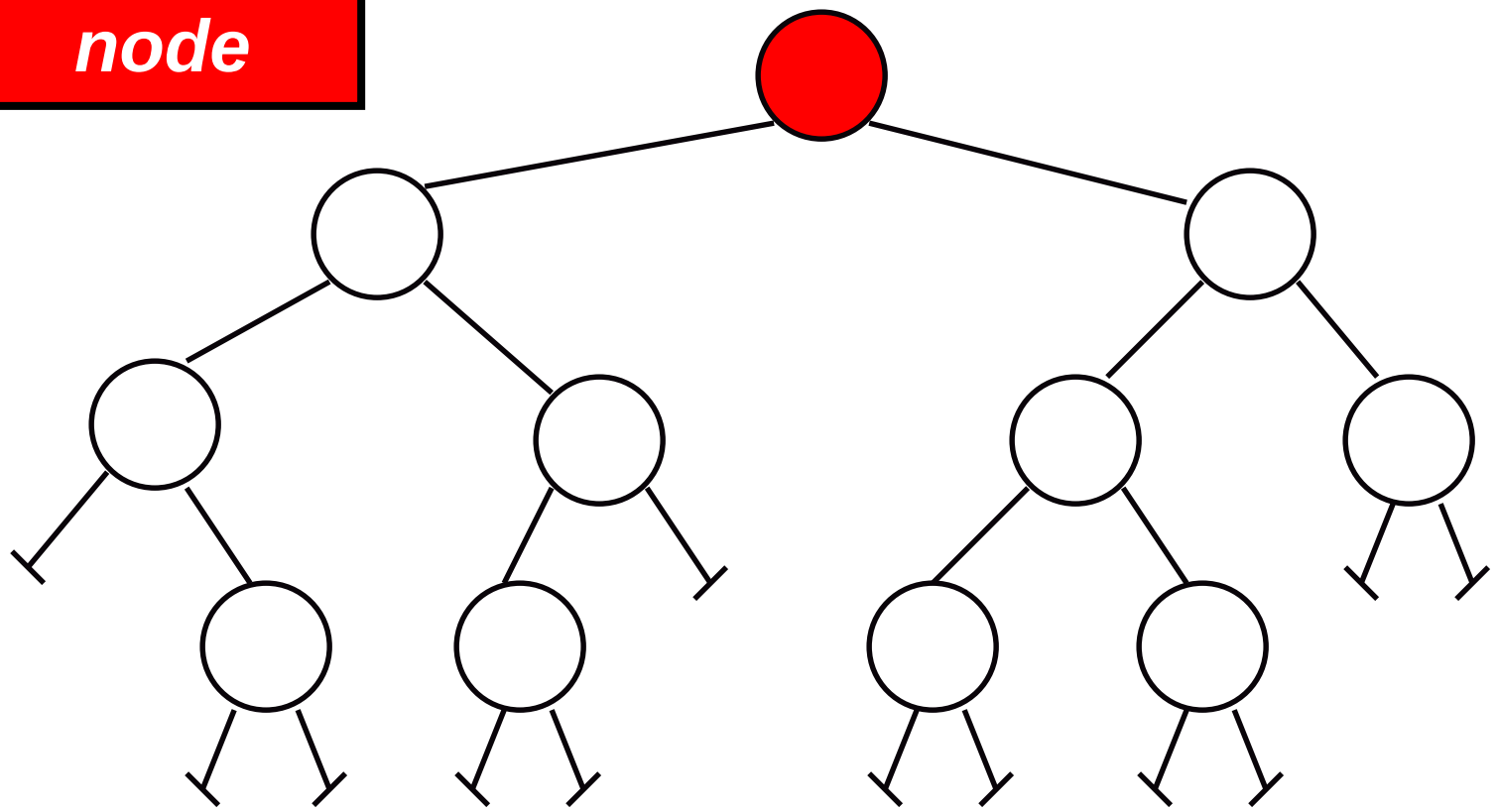


Parts of a Tree

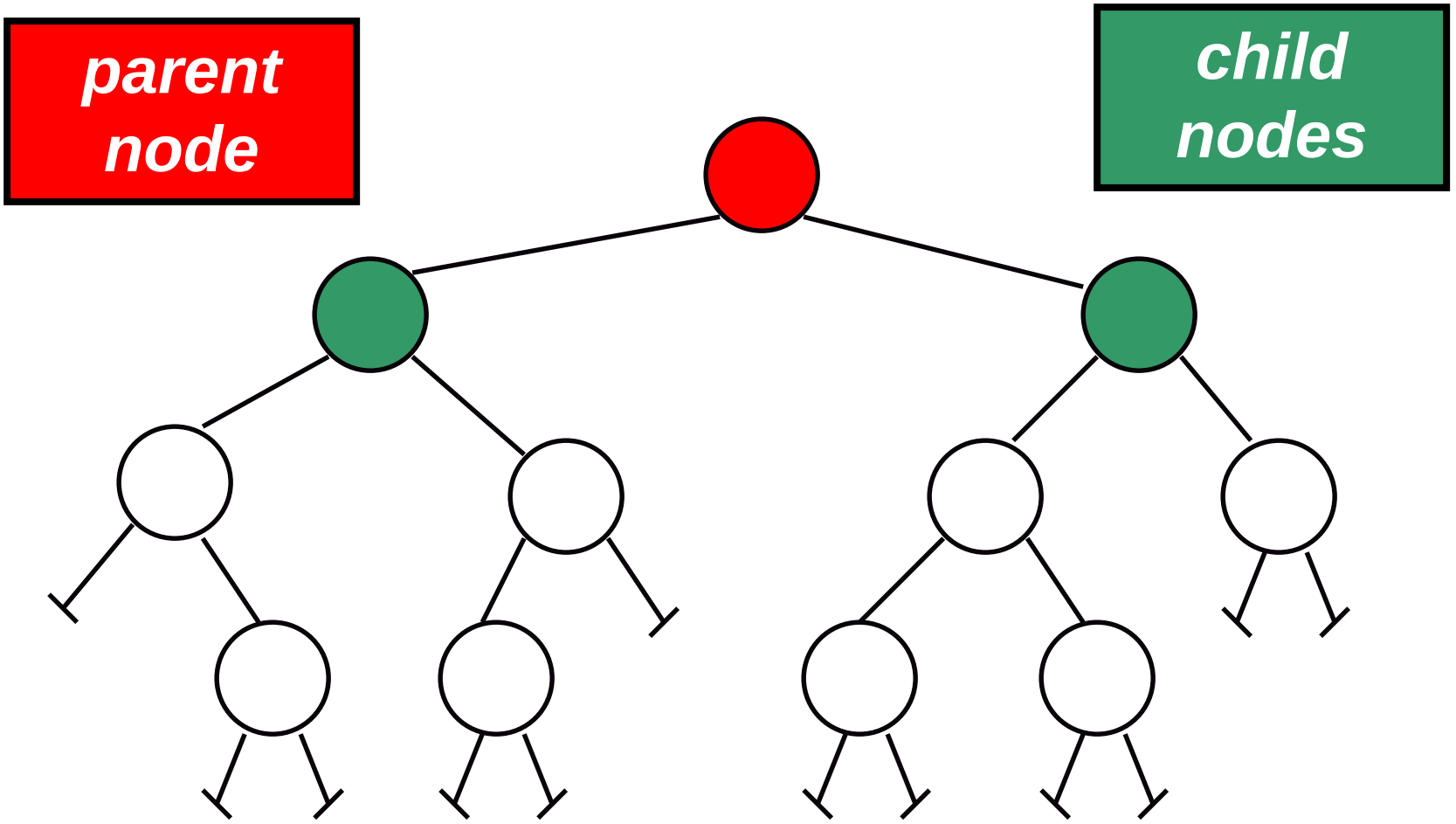


Parts of a Tree

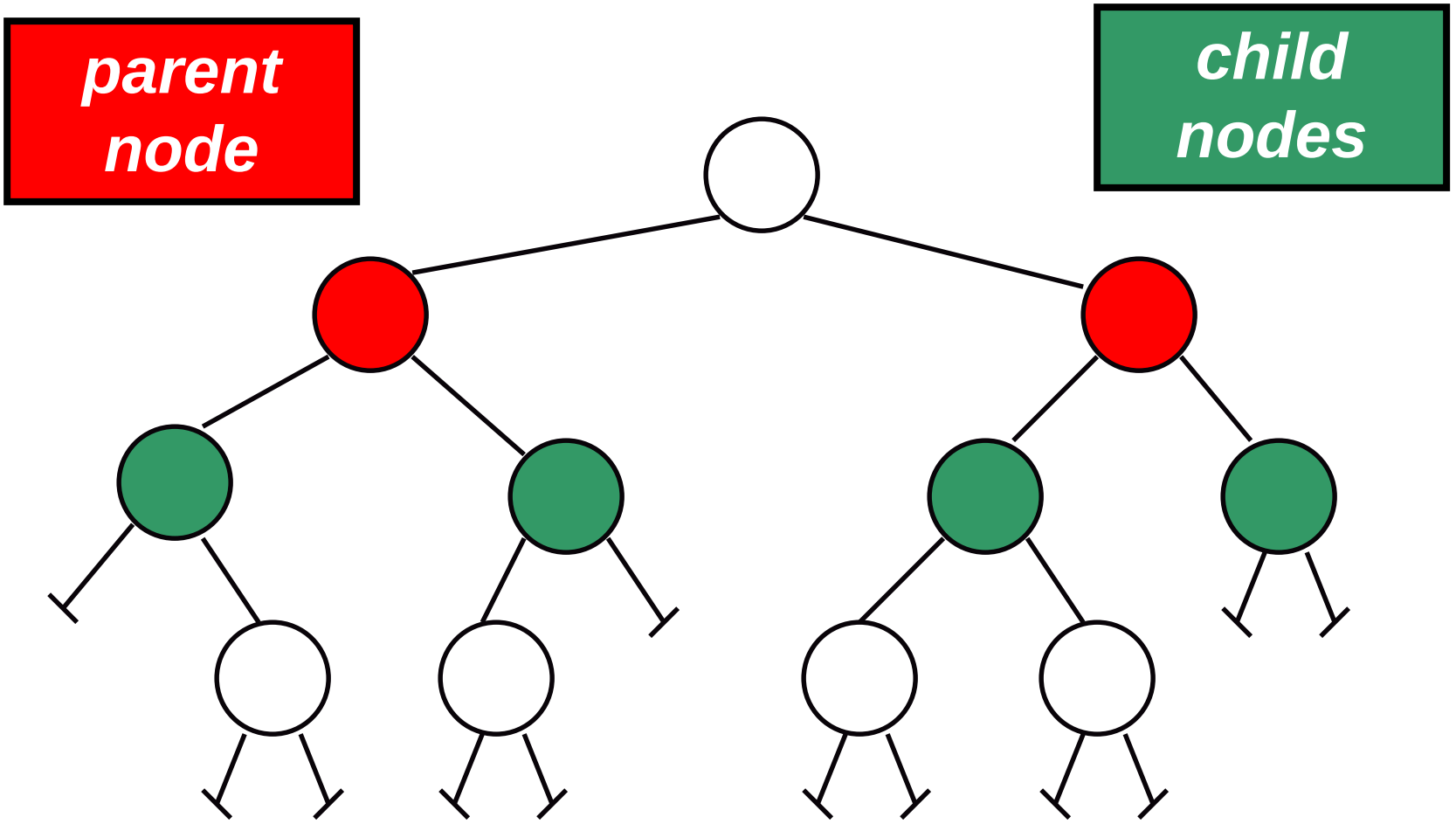
parent node



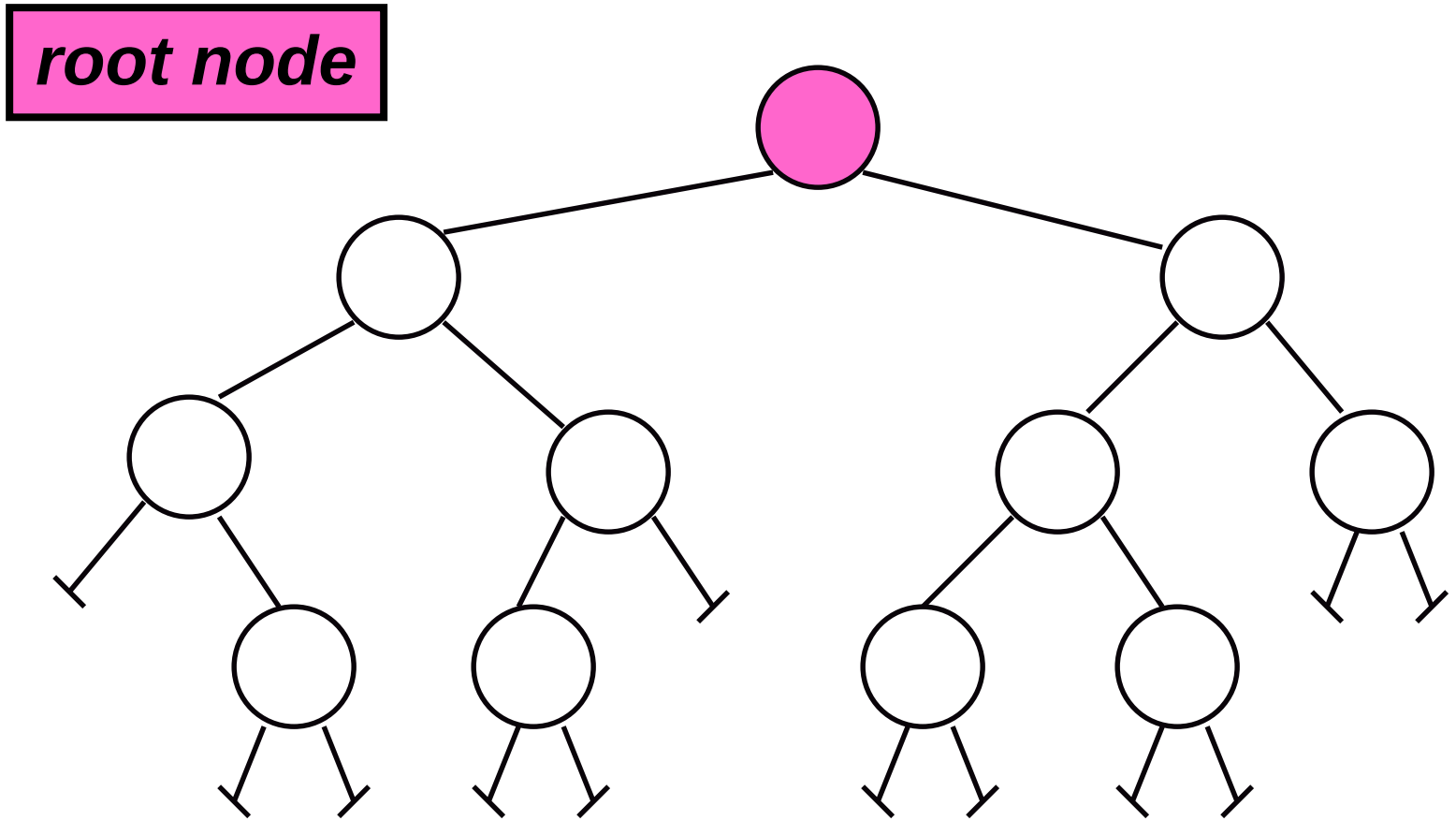
Parts of a Tree



Parts of a Tree

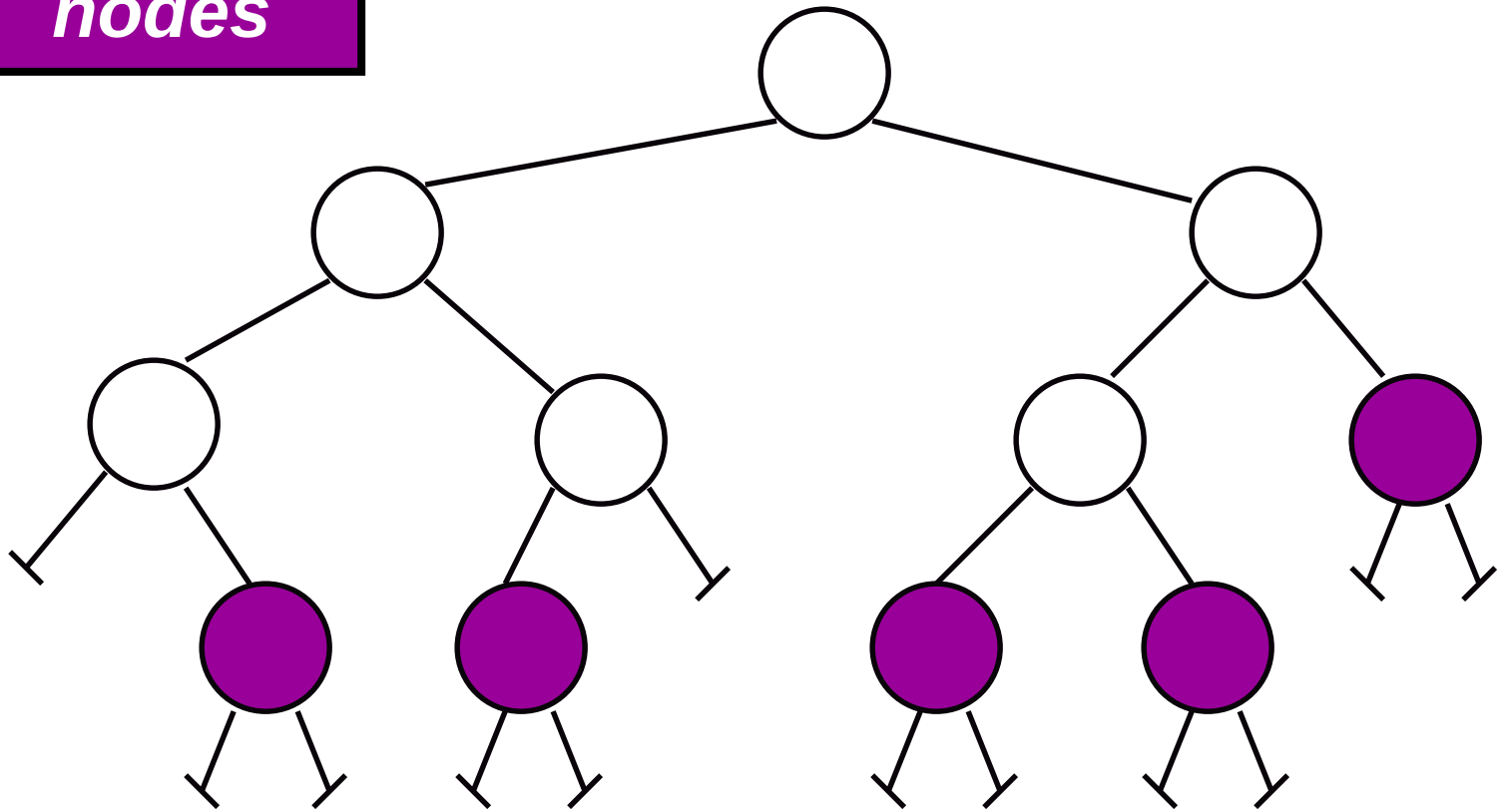


Parts of a Tree

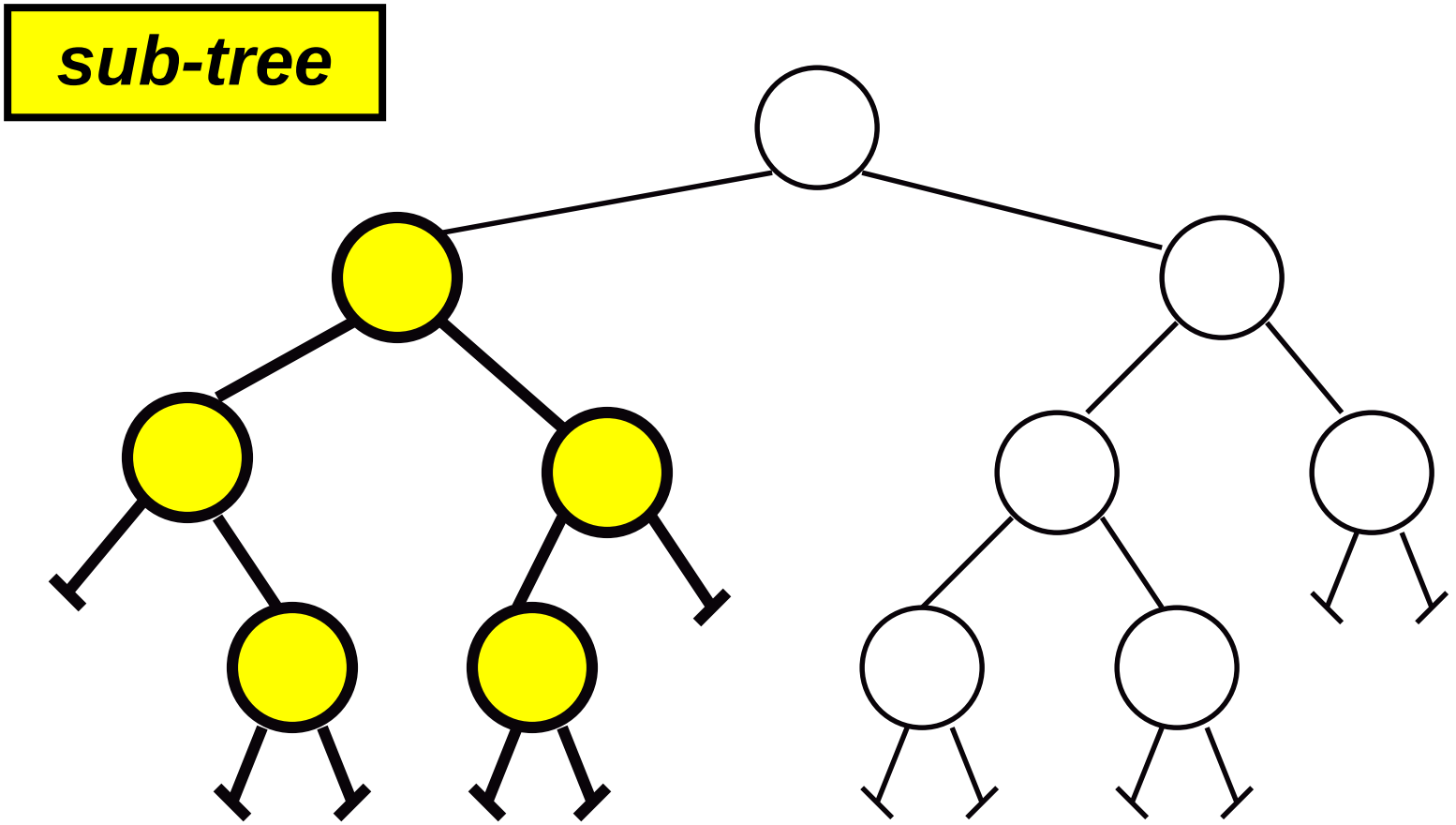


Parts of a Tree

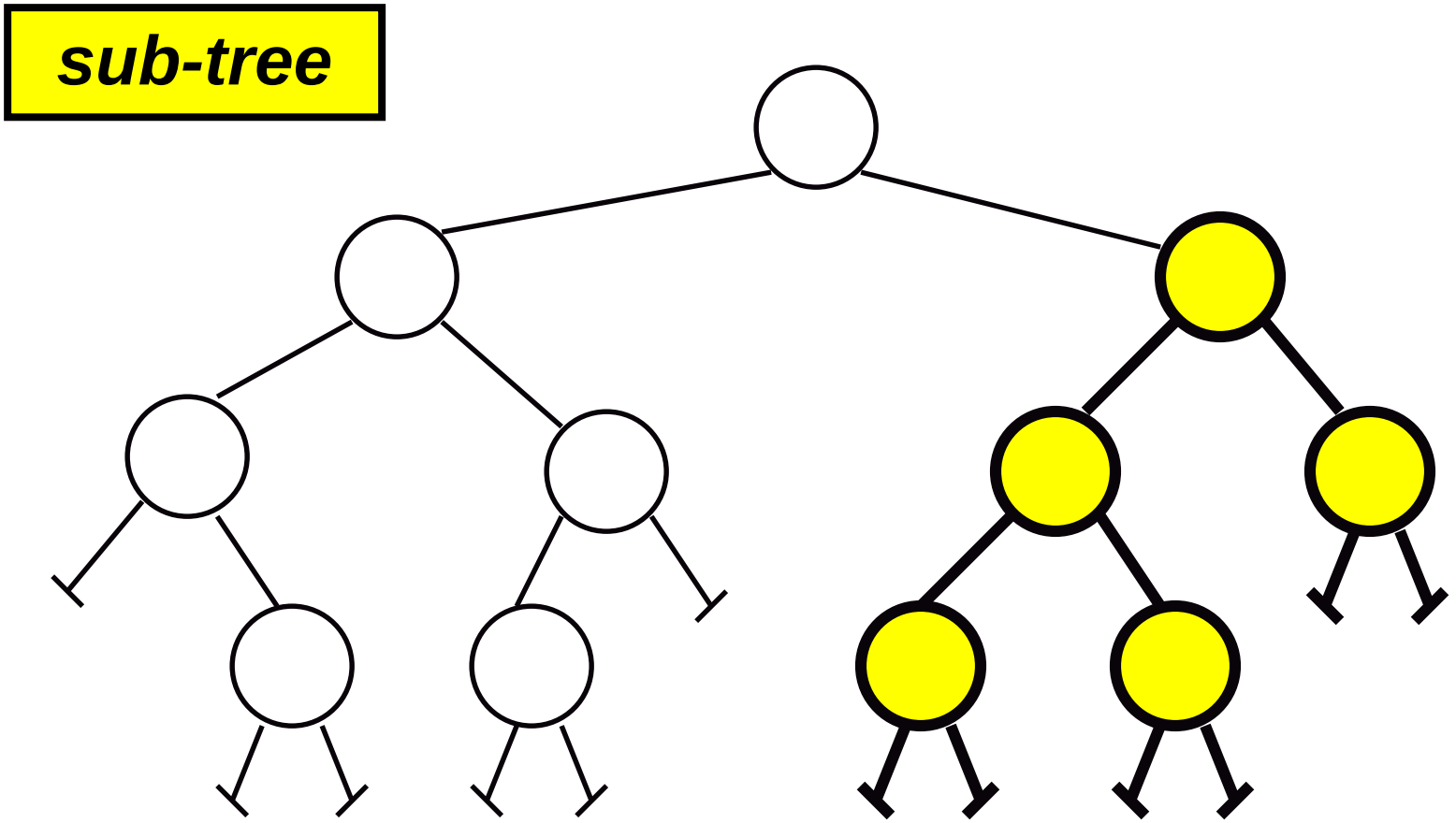
leaf nodes



Parts of a Tree

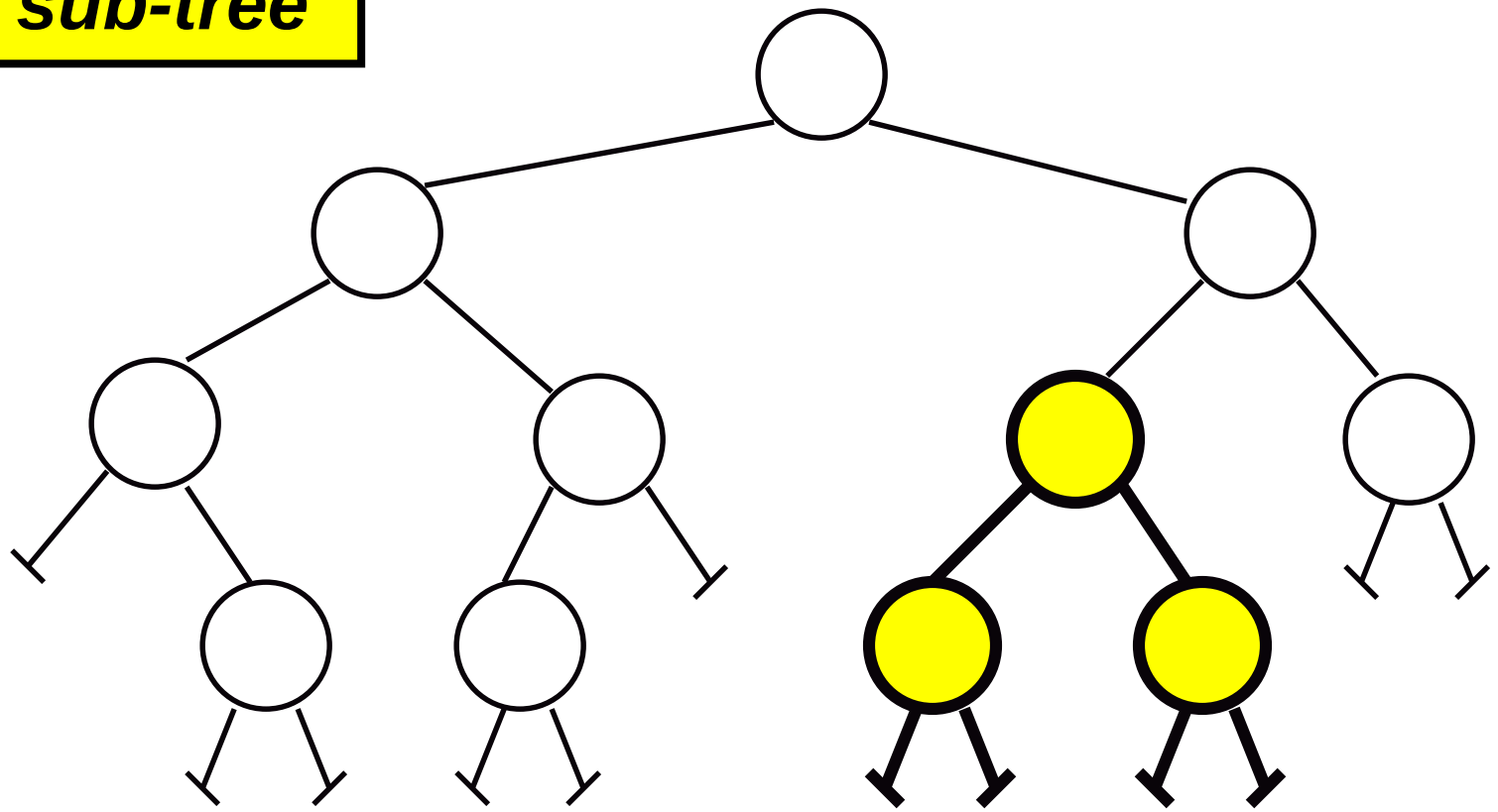


Parts of a Tree



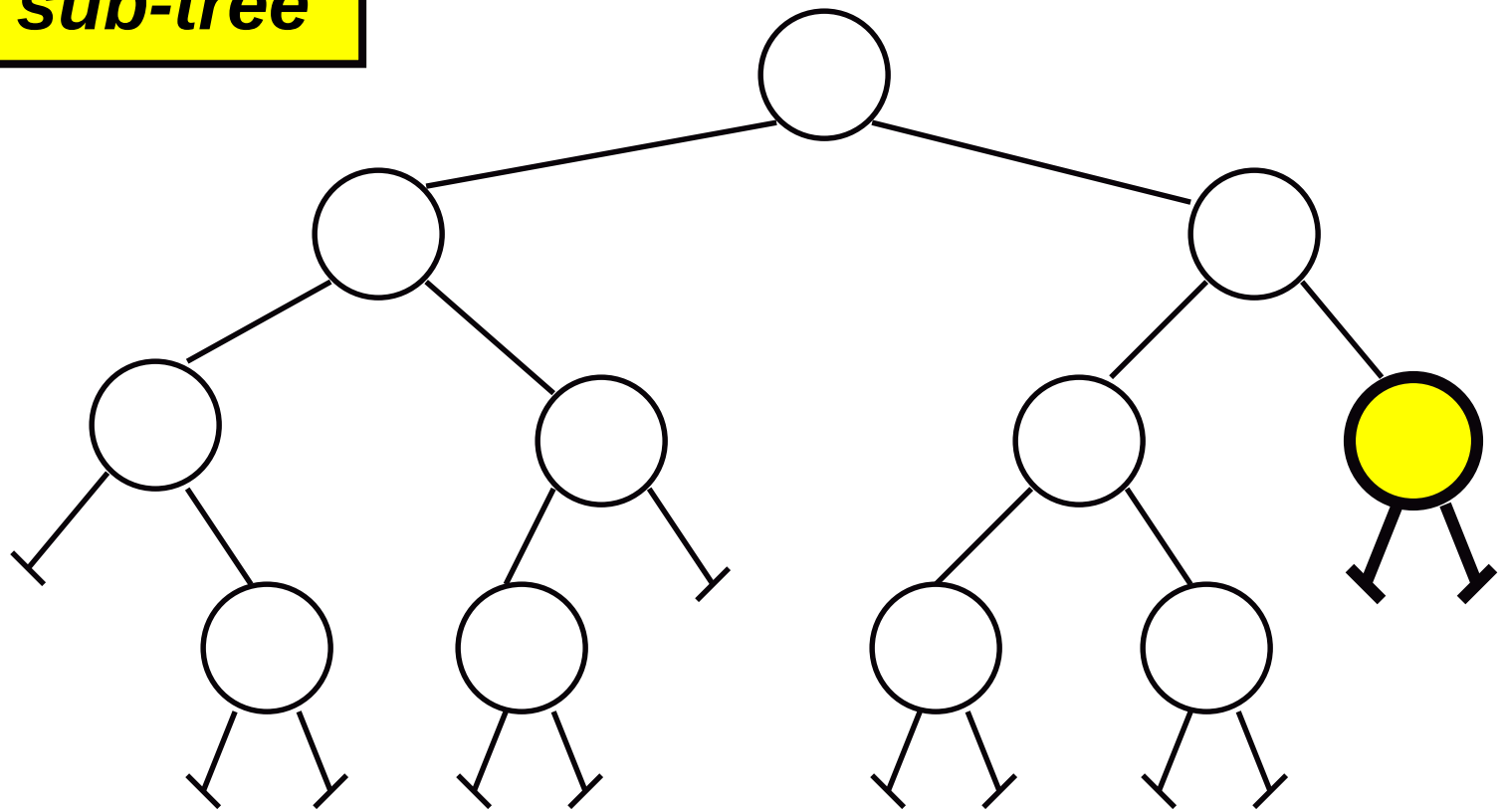
Parts of a Tree

sub-tree



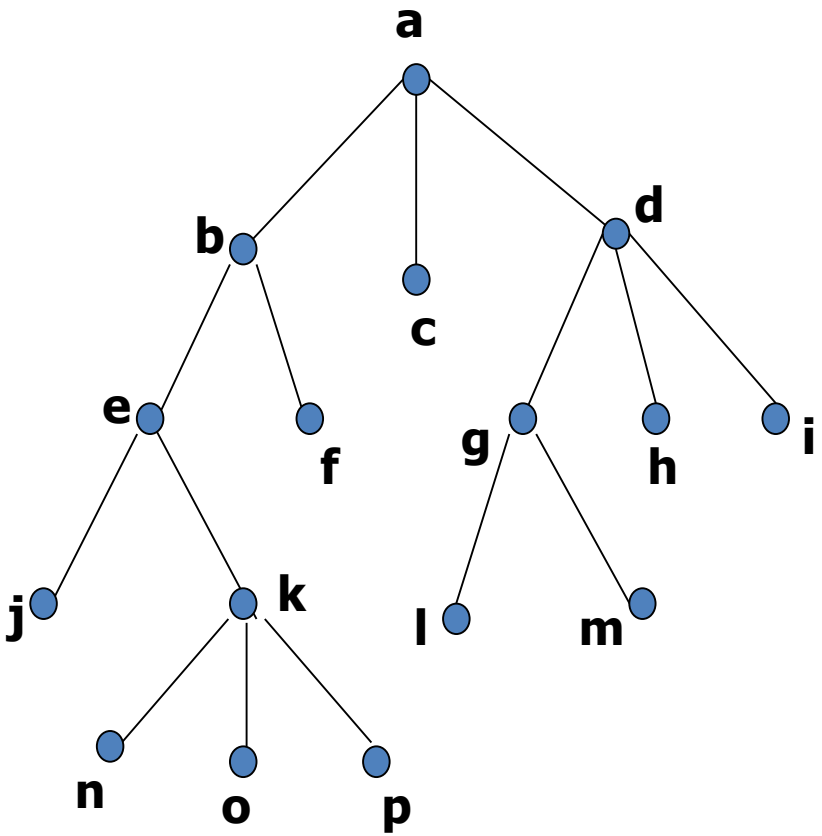
Parts of a Tree

sub-tree

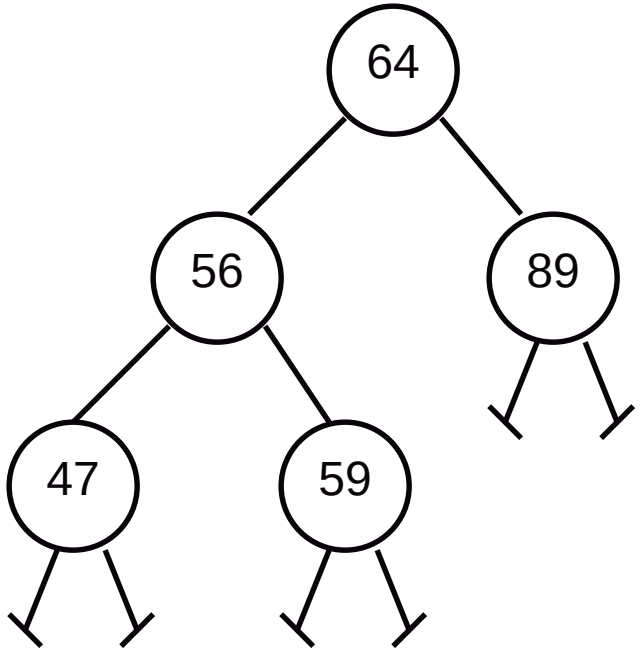


m-array Trees

A Tree is an m-array Tree when each of its node has no more than m children.



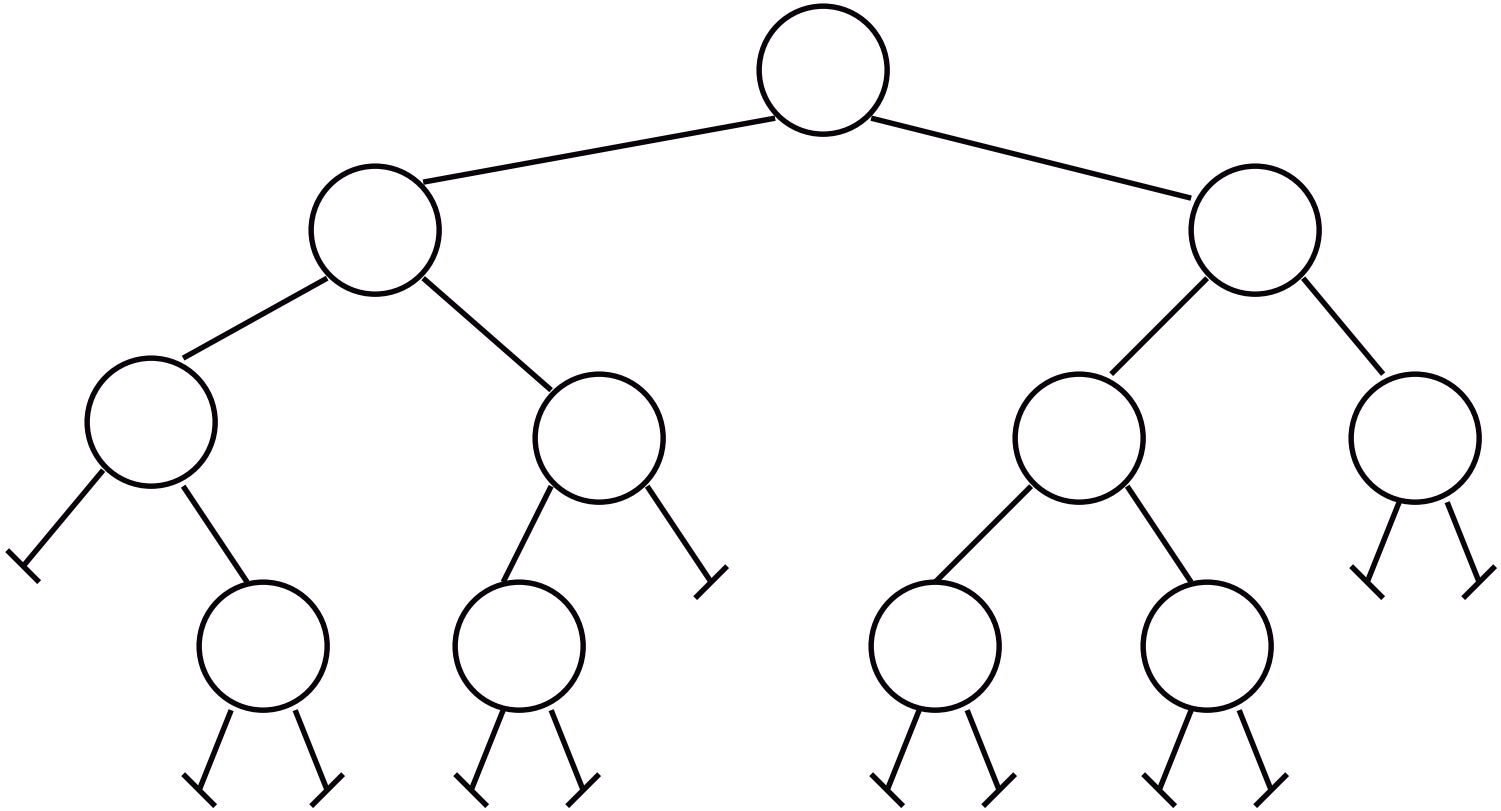
Three-array trees
(m=3)



Two-array trees
(m=2)

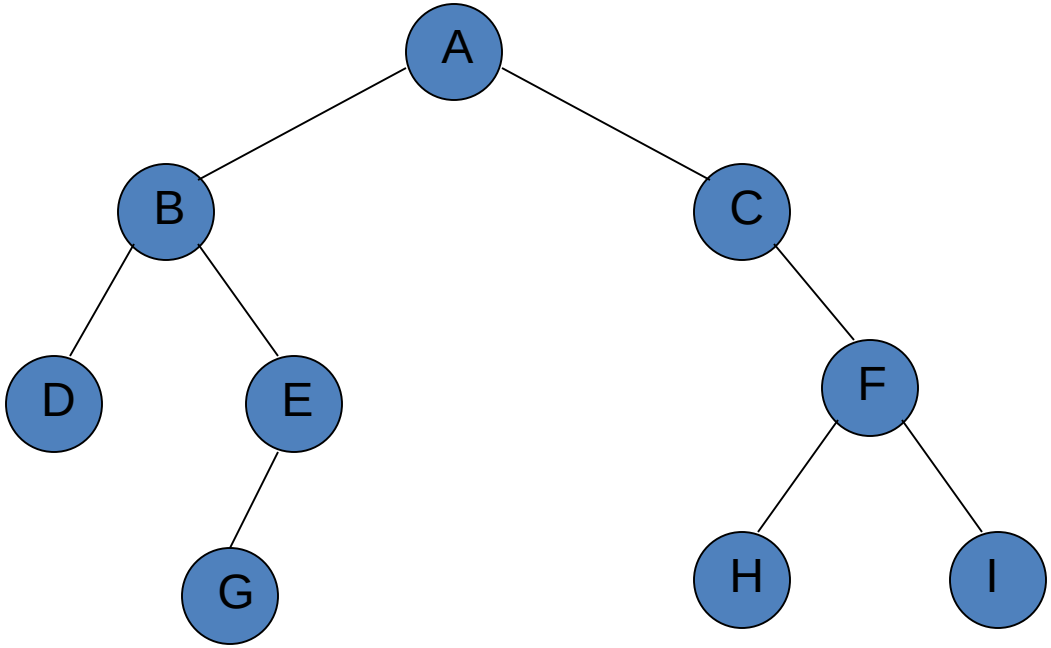
Binary Tree (BT)

- Each node can have at most 2 children

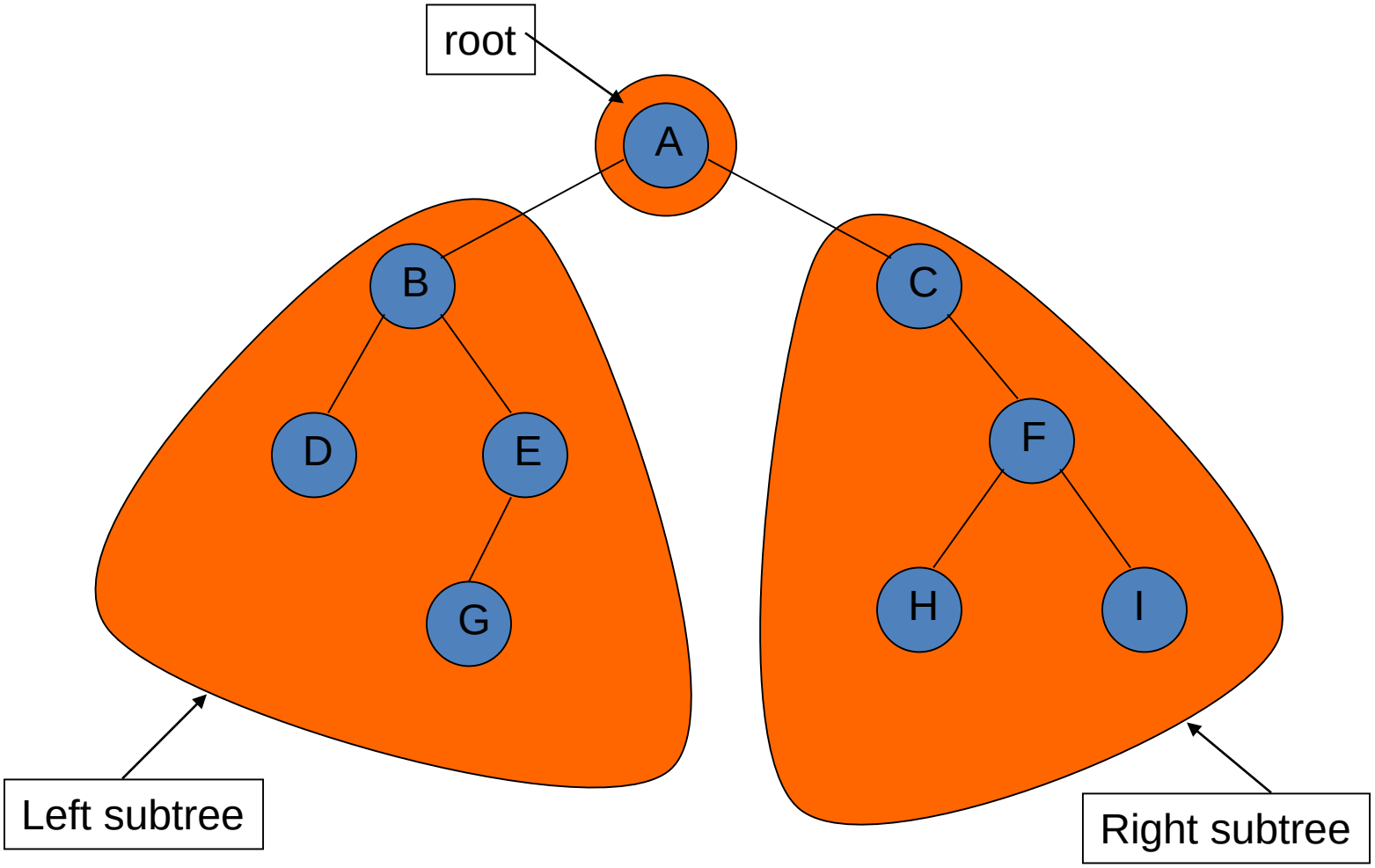


Binary Tree

- Binary tree with 9 nodes.

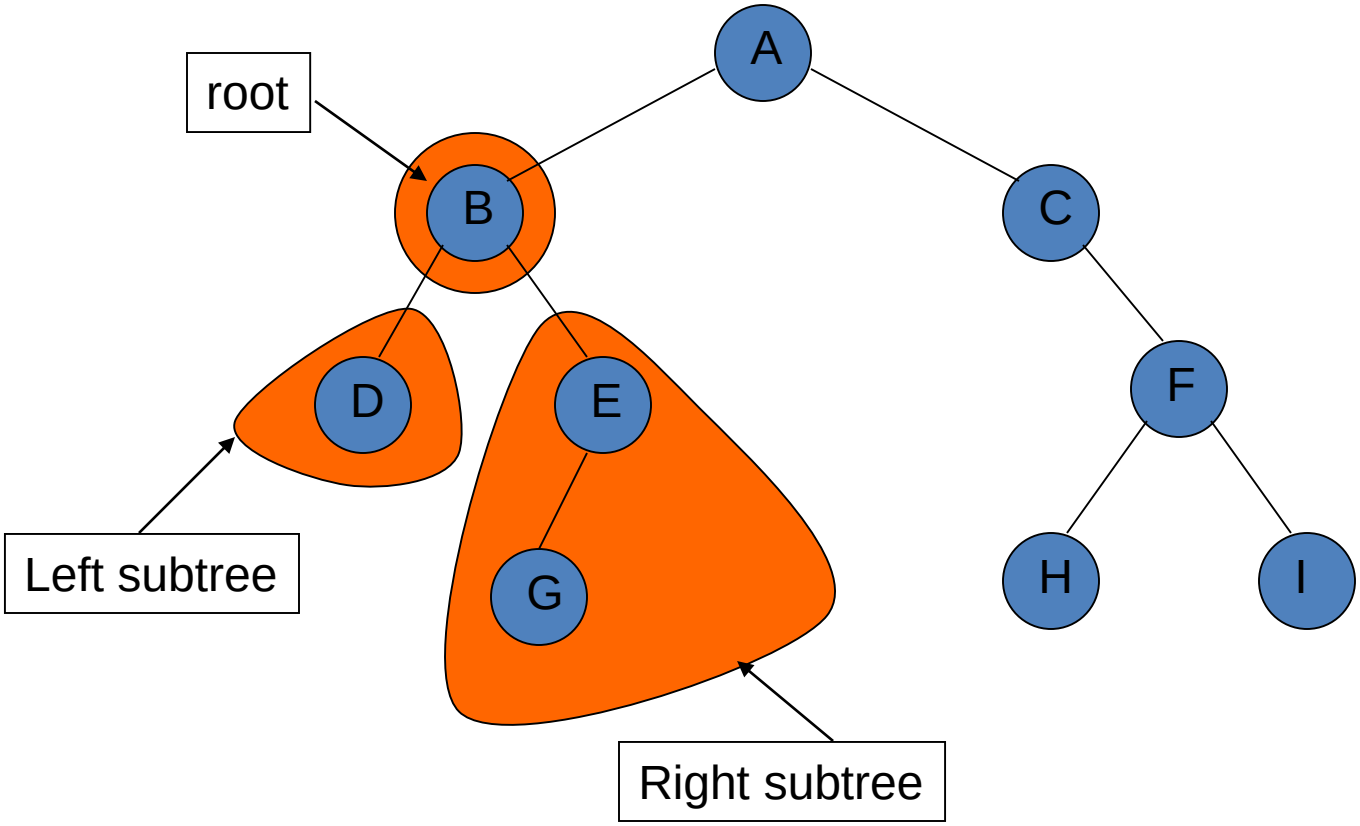


Binary Tree



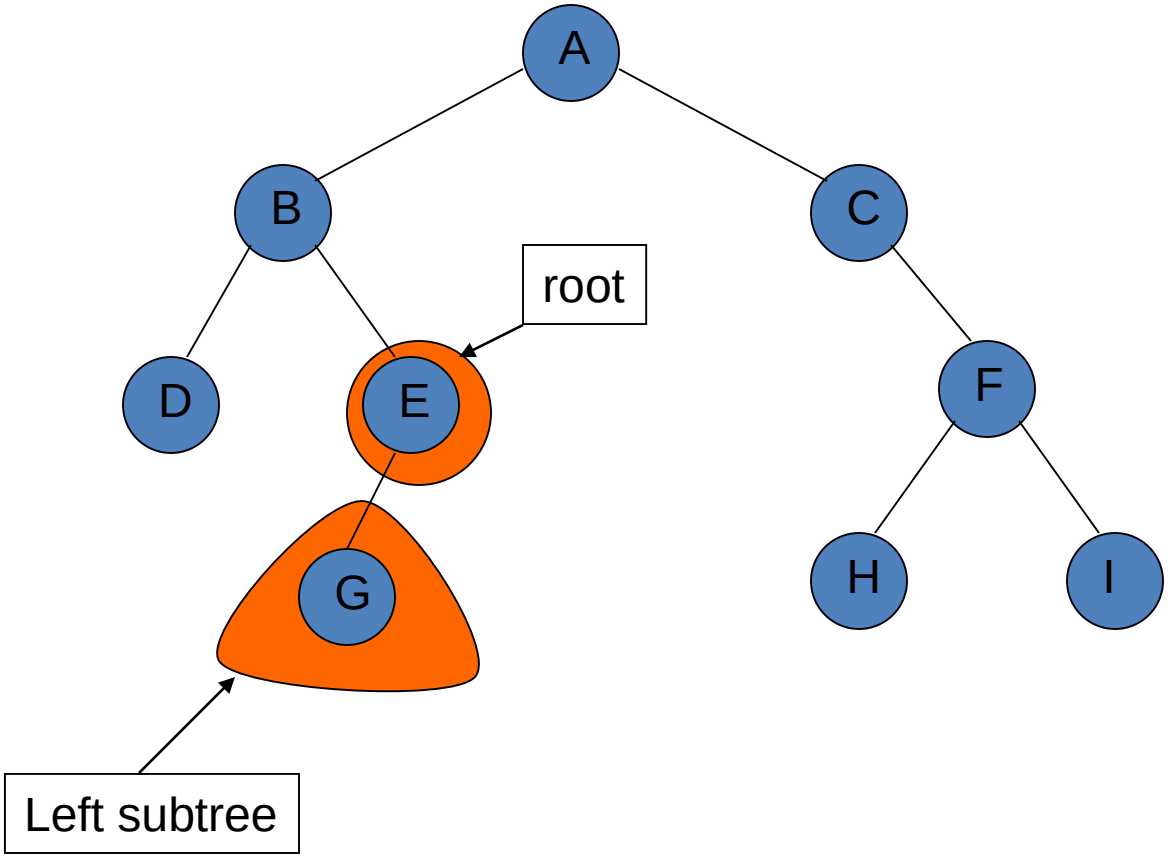
Binary Tree

- Recursive definition



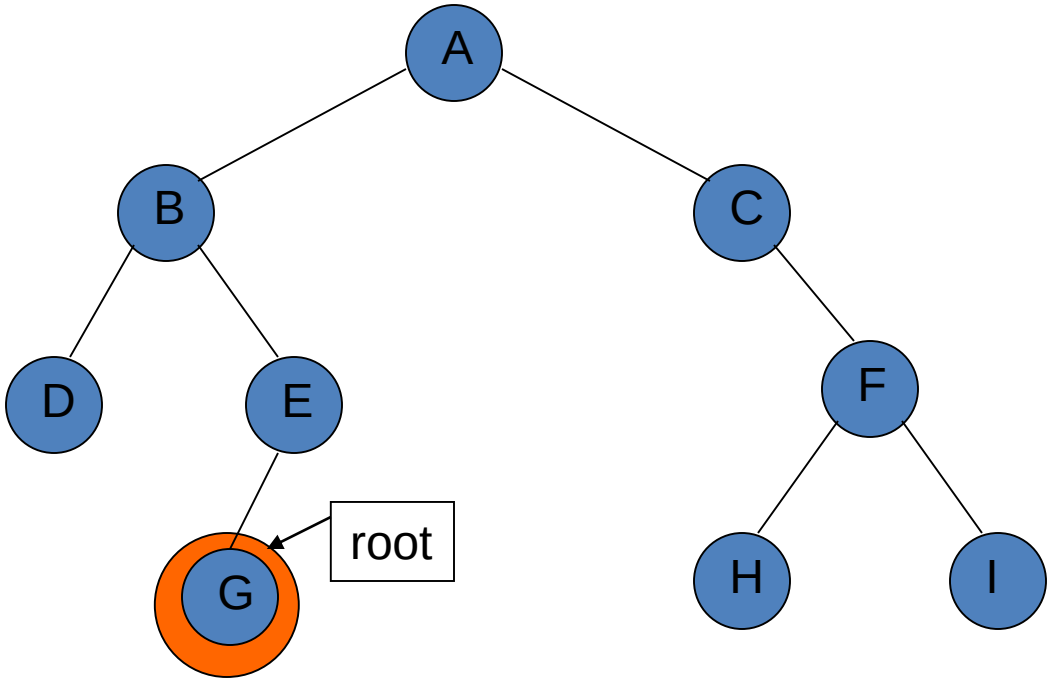
Binary Tree

- Recursive definition



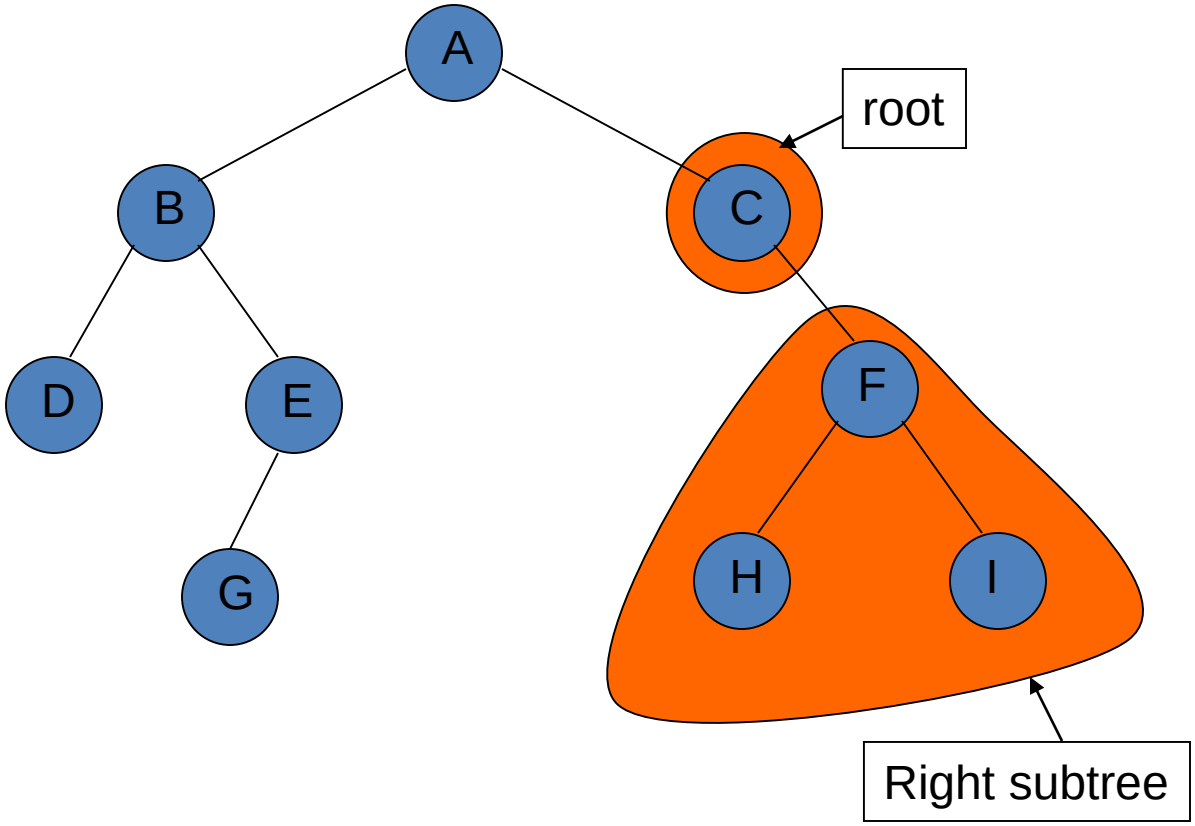
Binary Tree

- Recursive definition



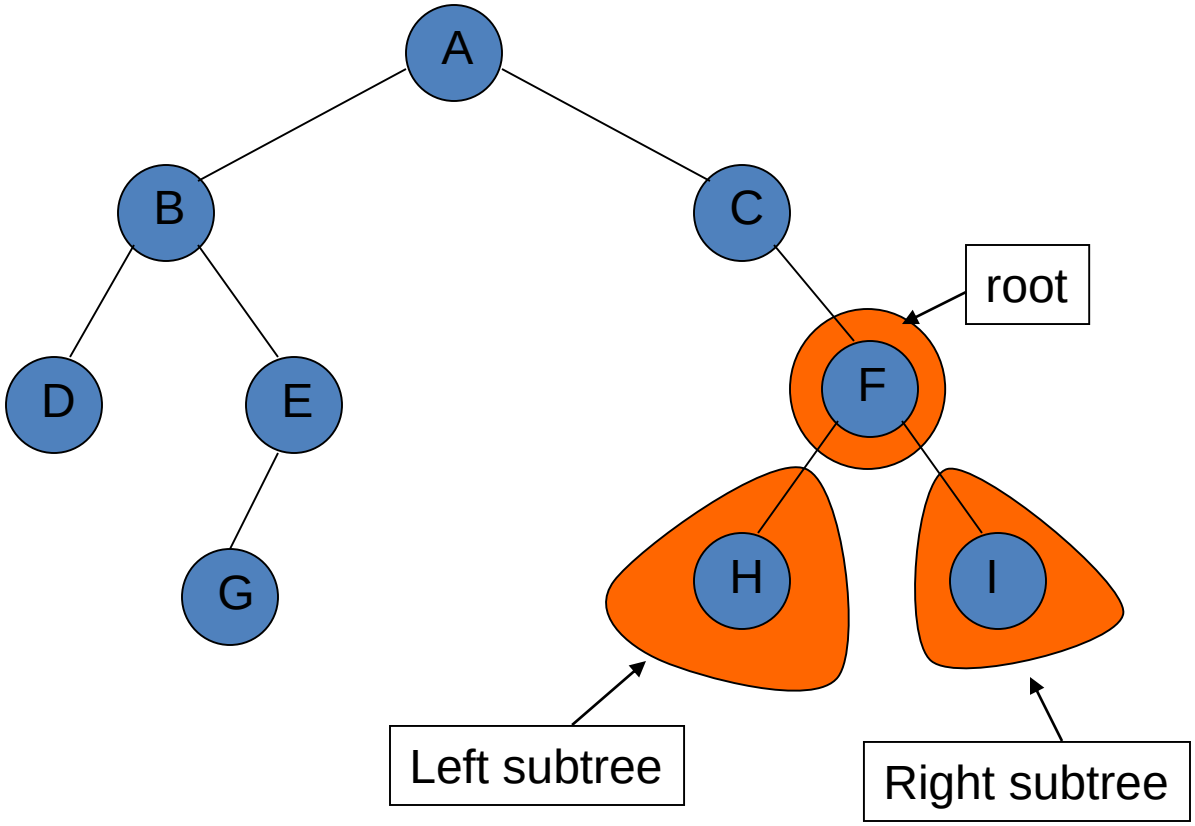
Binary Tree

- Recursive definition



Binary Tree

- Recursive definition

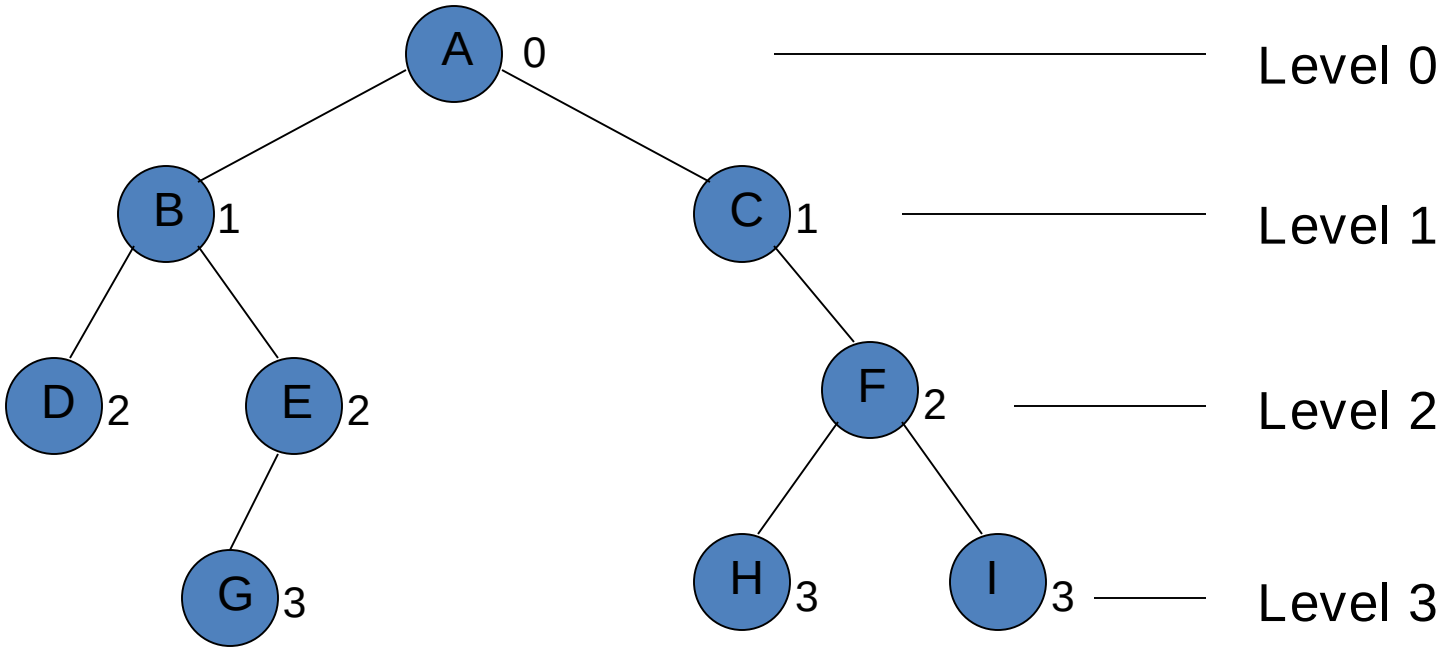


Level & Depth of a Binary Tree

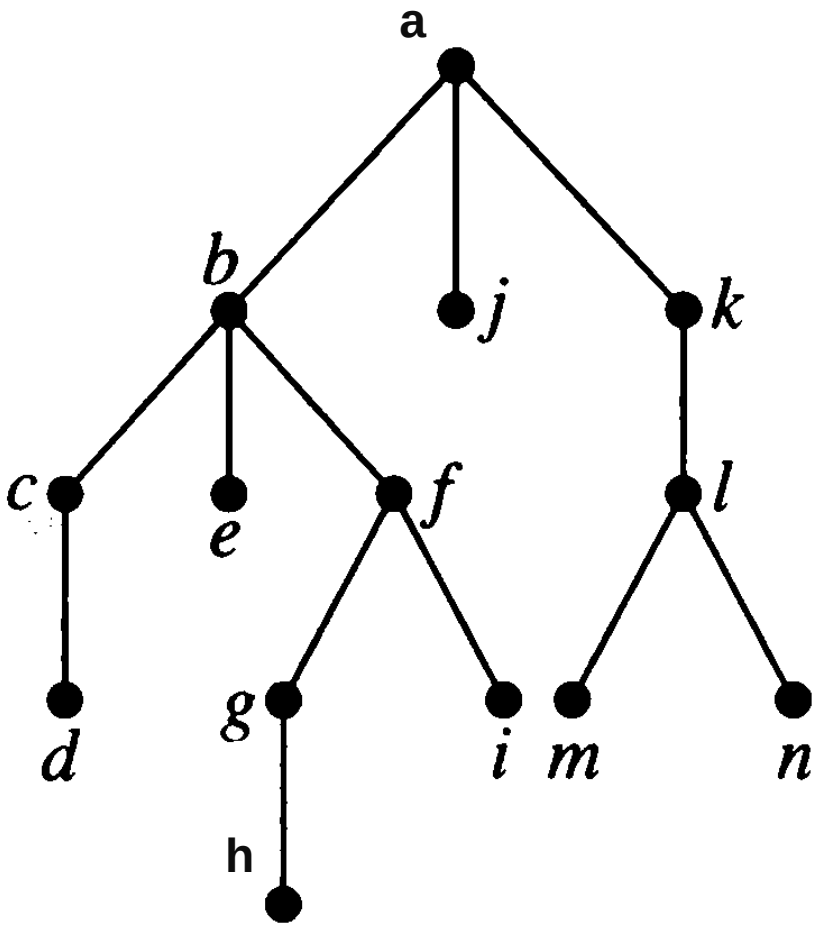
- The level of a node in a binary tree is defined as follows:
  - Root has level 0,
  - Level of any other node is one more than the level its parent (father).

- The depth of a tree means how many levels is in the tree i.e. the Total number of level in the tree.

Level of a Binary Tree Node



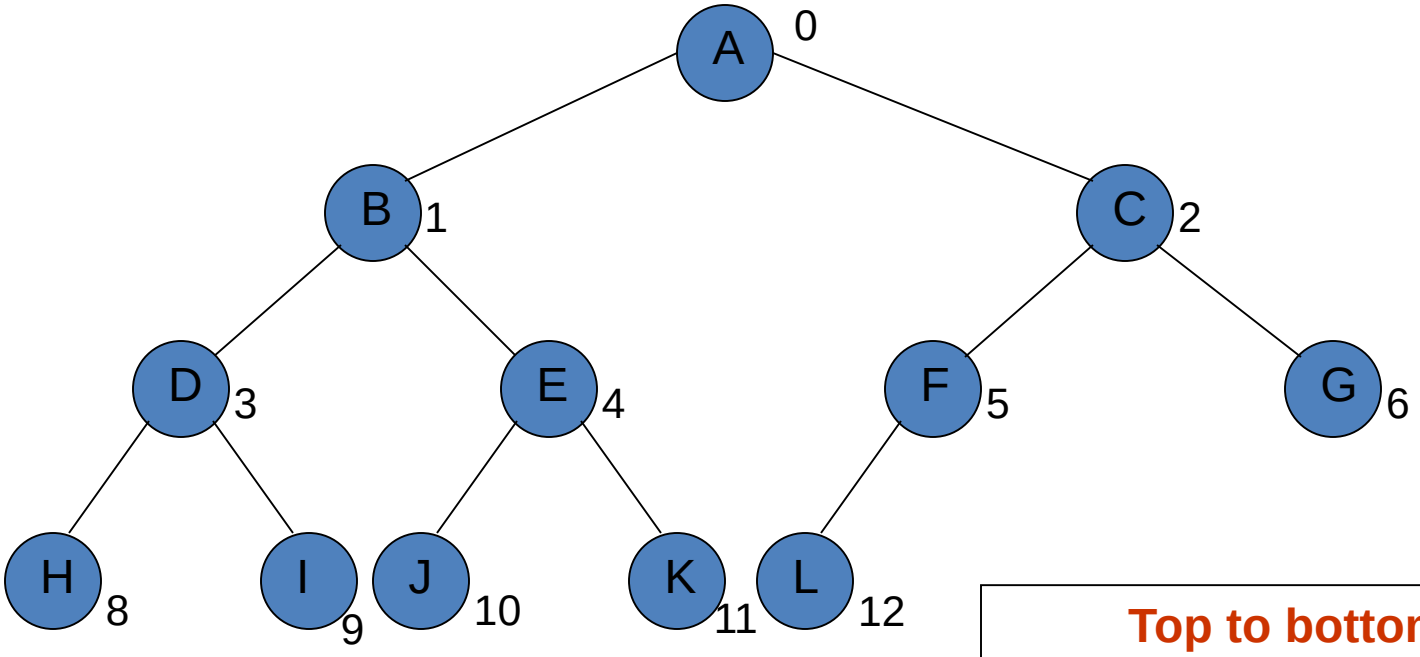
Find the level of each node in the tree shown in Figure.
What is the height/depth of this tree?



- The root a is at level O.
- Nodes b, j, and k are at level 1 .
- Nodes c, e, f, and I are at level 2.
- Nodes d, g, i , m , and n are at level 3 .
- Finally, Node h is at level 4.
- The height/depth of this tree is 5.
- The height of node h is 0.
- The height of node b is 3.
- The depth of node m is 3.

Complete Binary Tree

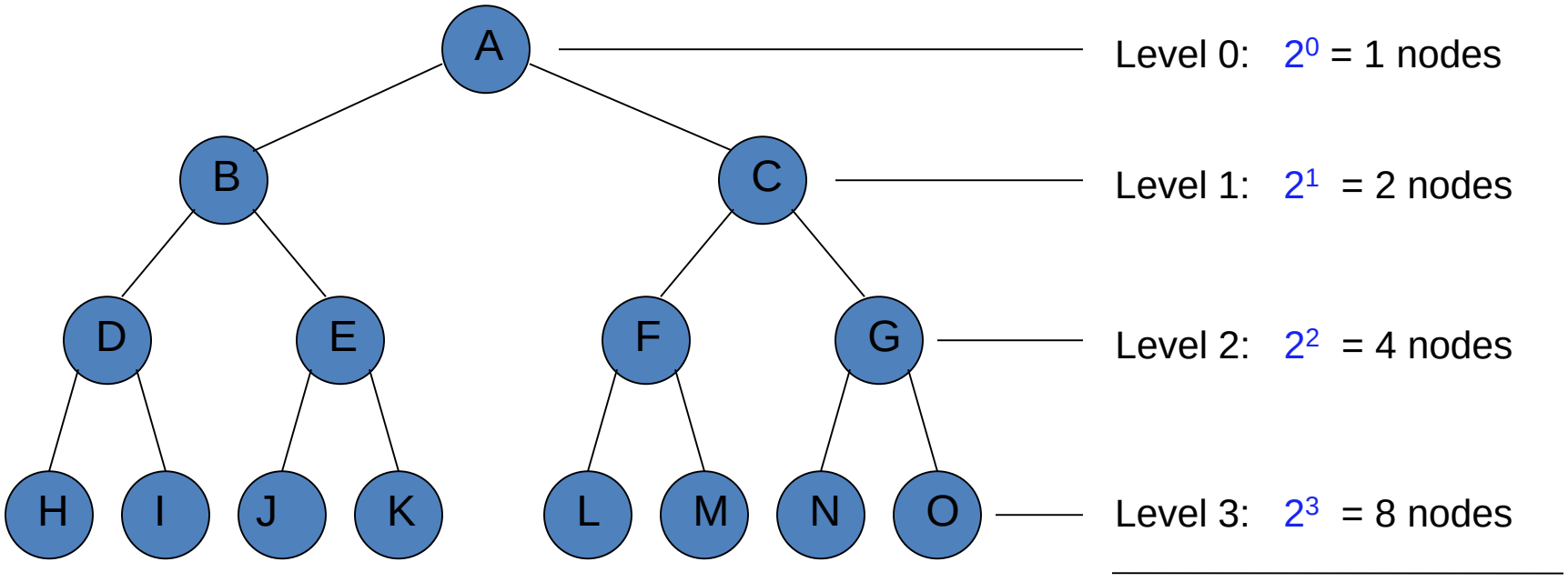
A complete binary tree is a binary tree, which is completely filled, with the possible exception of the bottom level, which is filled from left to right.



Top to bottom,
From left to right number

Full Binary Tree

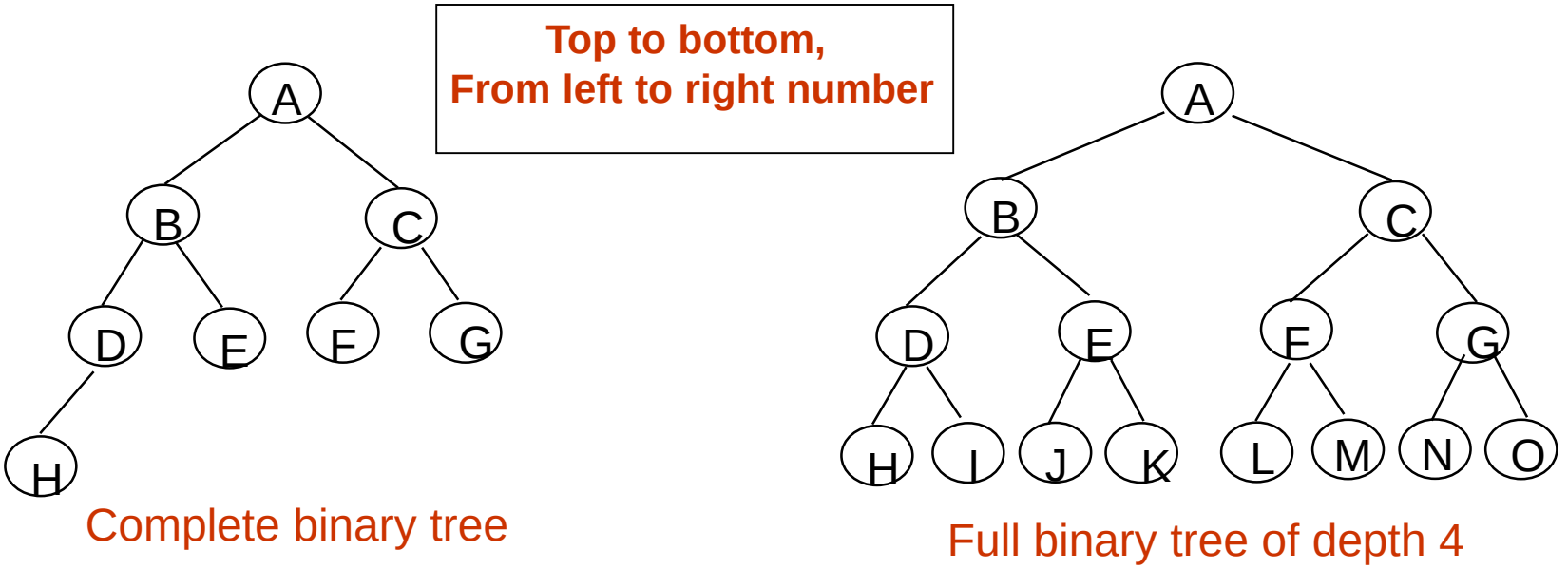
- A full binary tree of depth *k* is a binary tree of depth *k* having $2^k - 1$ nodes, *k*>=0.



Here depth is **k = 4** and Total nodes **(2^4 -1)** = 15

Full BT VS Complete BT

- A full binary tree of depth *k* is a binary tree of depth *k* having 2^k-1 nodes, *k*>=0.
- A binary tree with *n* nodes and depth *k* is complete *iff* its nodes correspond to the nodes numbered from 0 to *n-1* in the full binary tree of depth *k*..

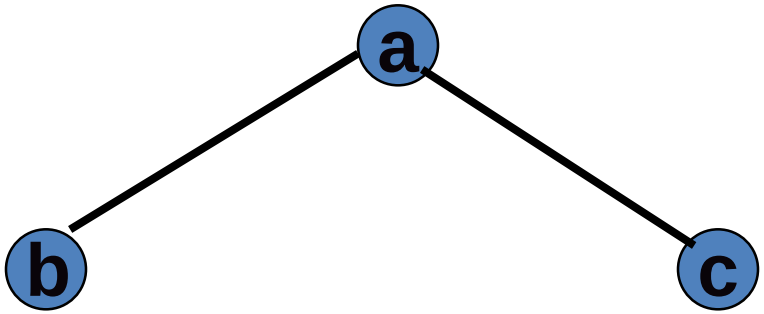


Traversal

- **Systematic way of visiting all the nodes.**
- **Methods:**
  - **Inorder**
  - **Postorder**
  - **Preorder**
- **They all traverse the left subtree before the right subtree.**
- **The name of the traversal method depends on when the node is visited.**

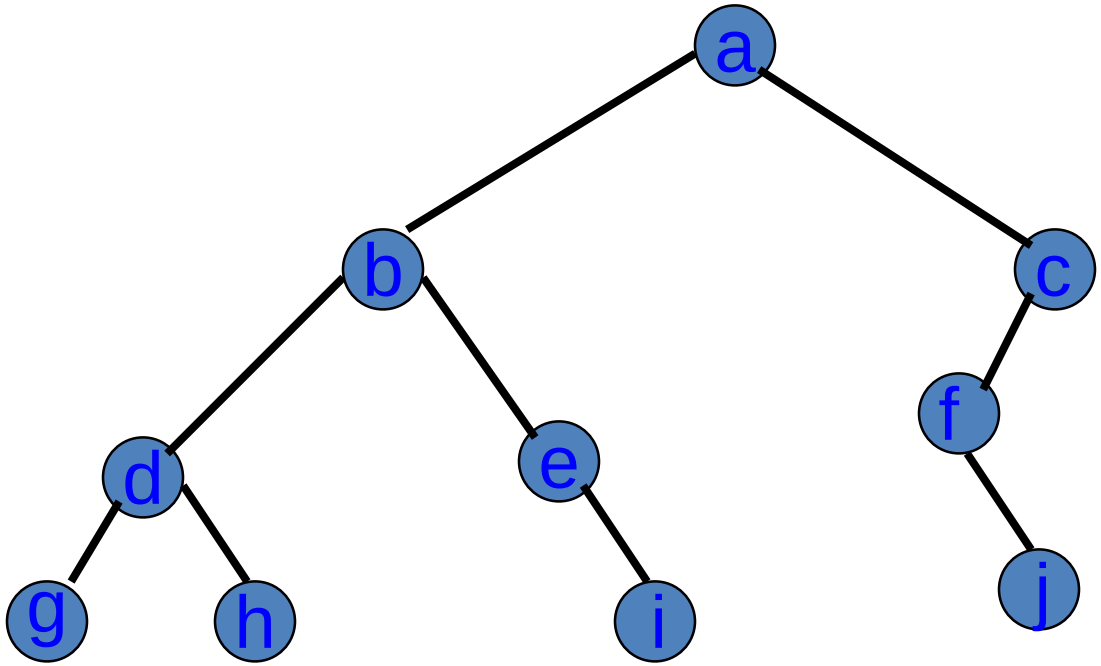
Inorder Traversal (Left_Node_Right)

- Traverse the left subtree.
- Visit the node.
- Traverse the right subtree.



b a c

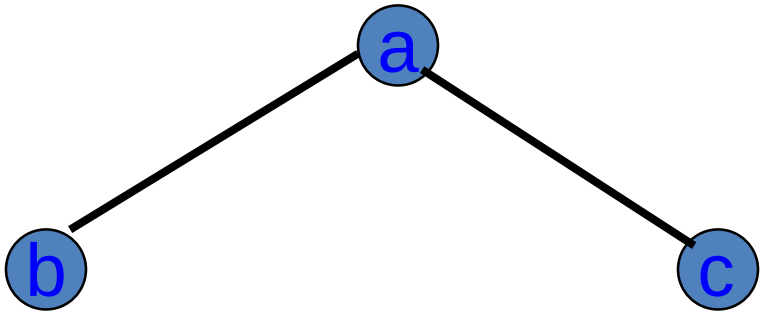
Inorder Example



g d h b e i a f j c

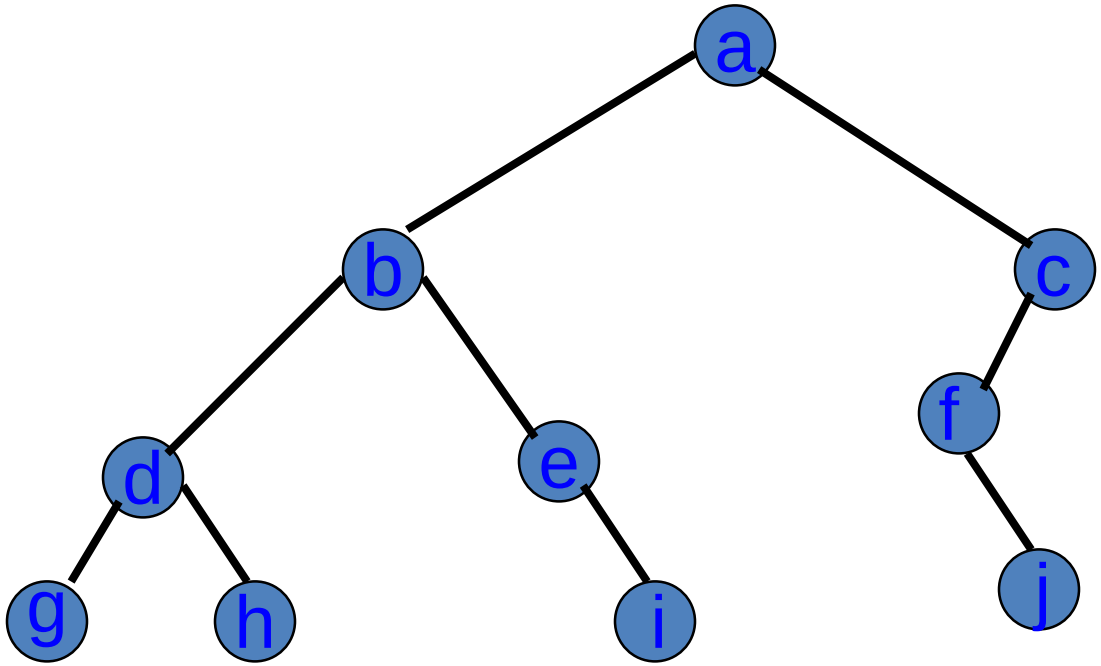
Postorder Traversal (Left_Right_Node)

- Traverse the left subtree.
- Traverse the right subtree.
- Visit the node.



b c a

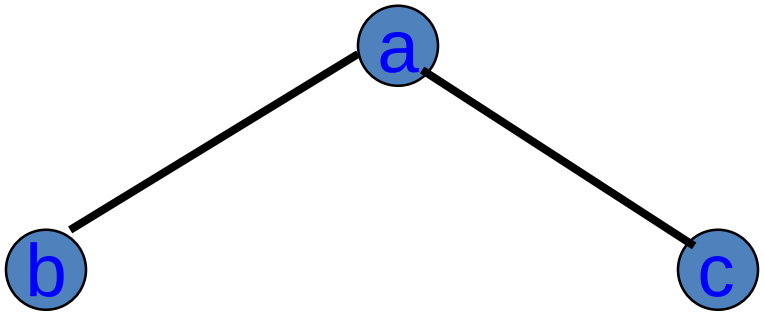
Postorder Example



g h d i e b j f c a

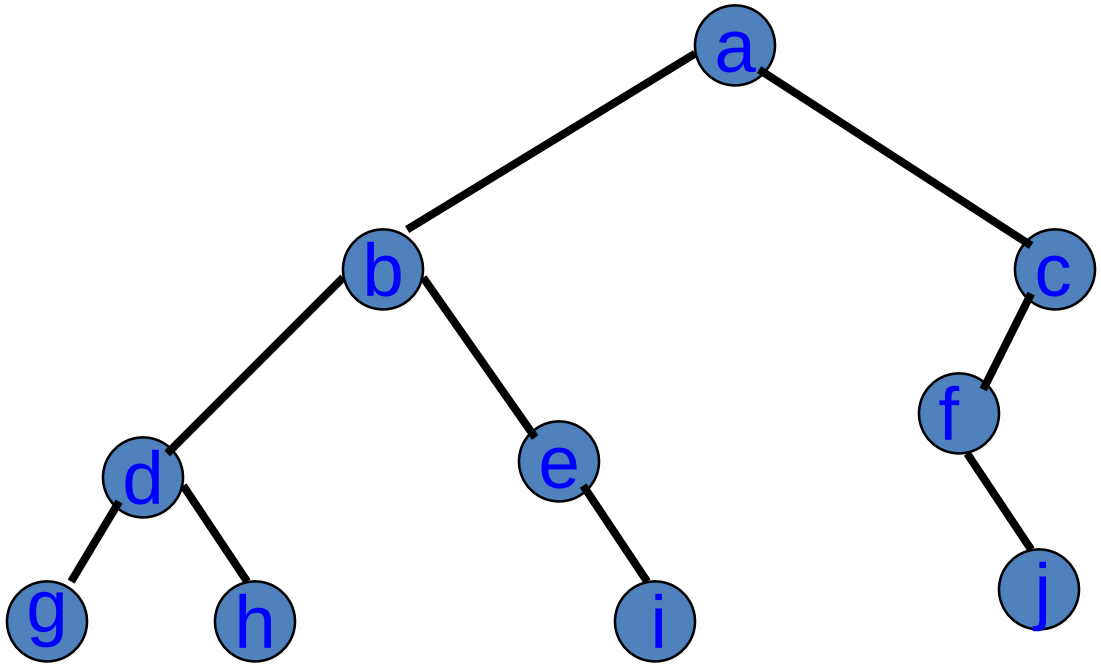
Preorder Traversal (Node_Left_Right)

- Visit the node.
- Traverse the left subtree.
- Traverse the right subtree.



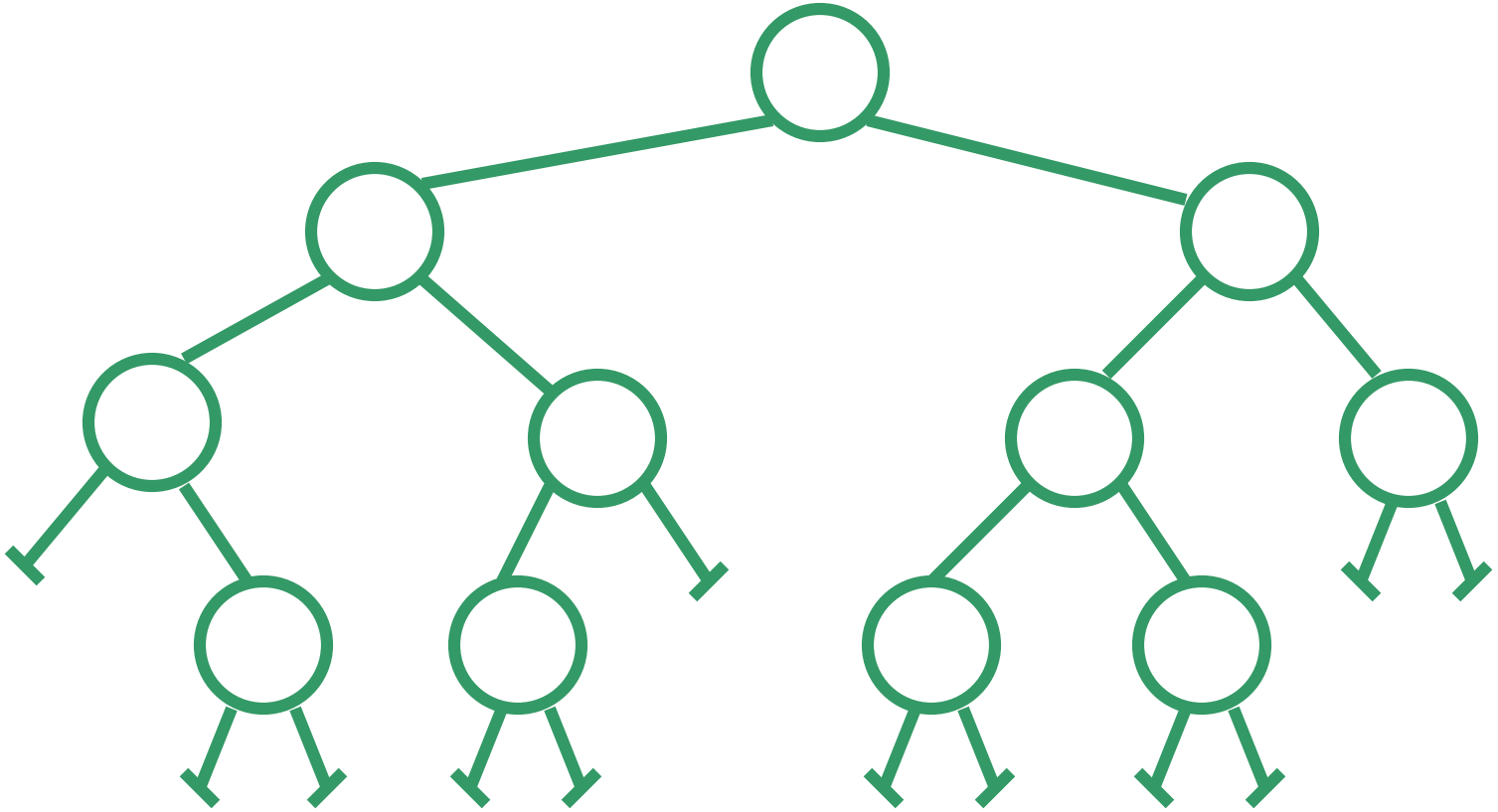
a b c

Preorder Example



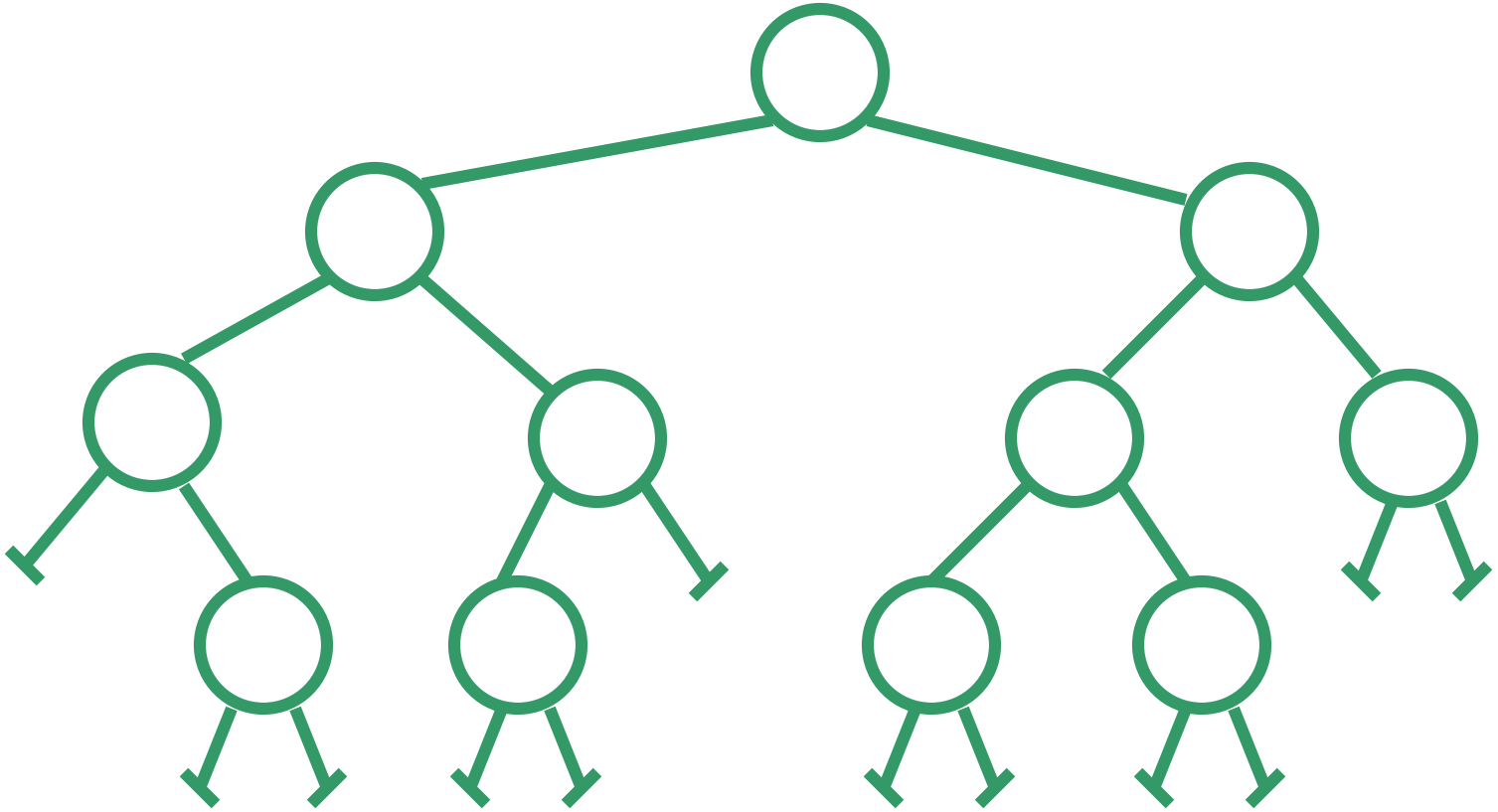
a b d g h e i c f j

Example: Inorder



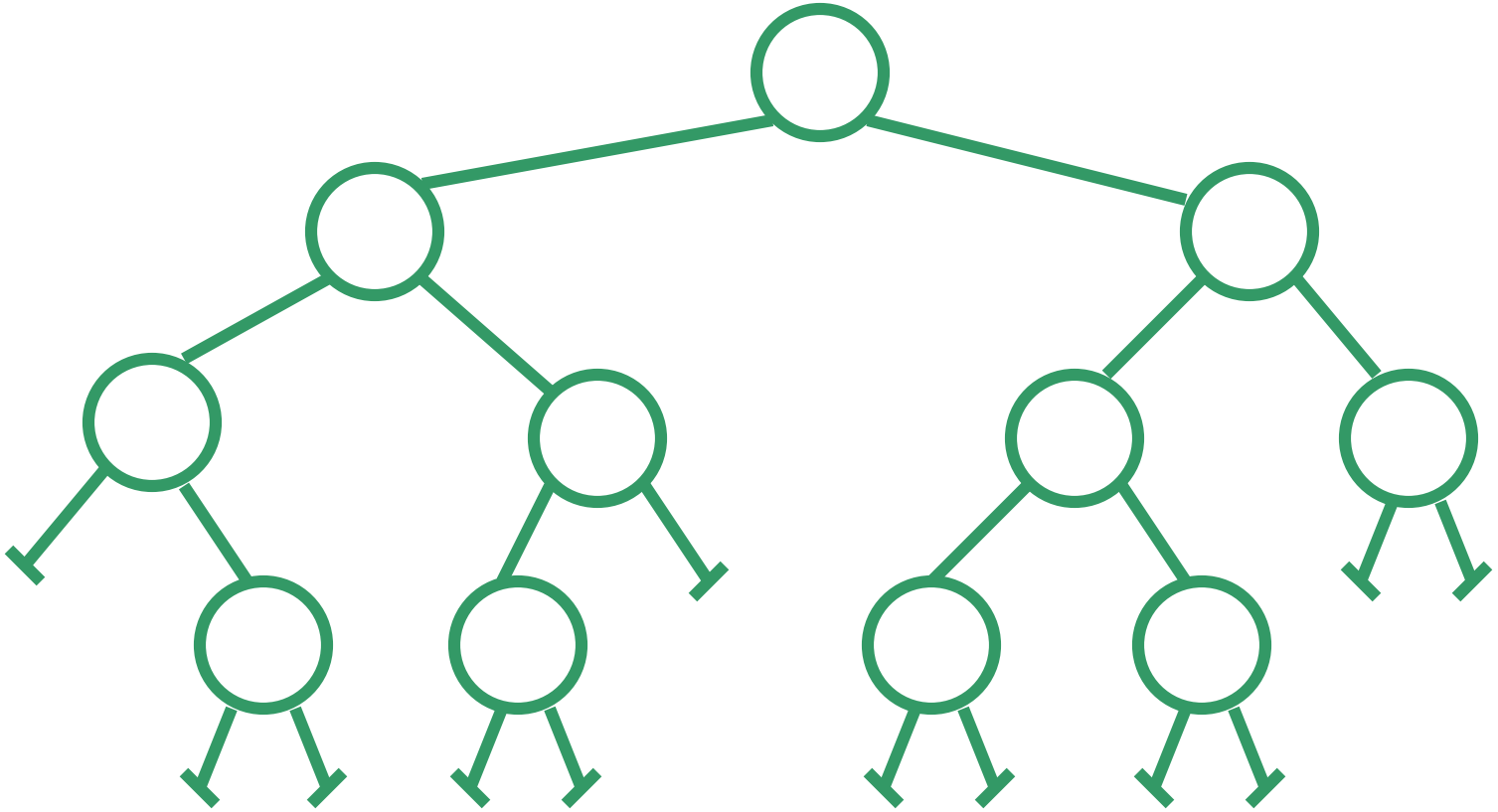
20	28	31	33	40	43	47	56	59	64	89

Example: Postorder



28	20	33	40	31	47	59	56	89	64	43

Example: Preorder

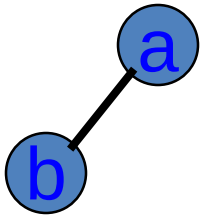


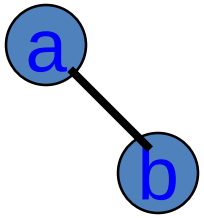
43	31	20	28	40	33	64	56	47	59	89

Some Examples

preorder

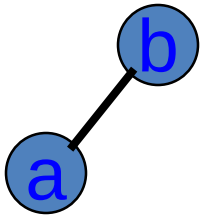
= ab

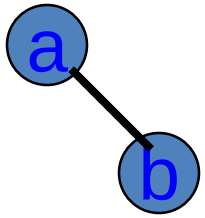




inorder

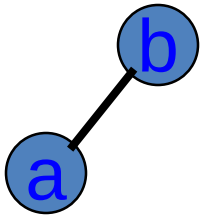
= ab

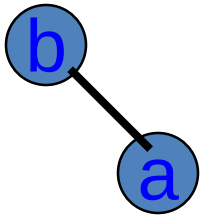




postorde

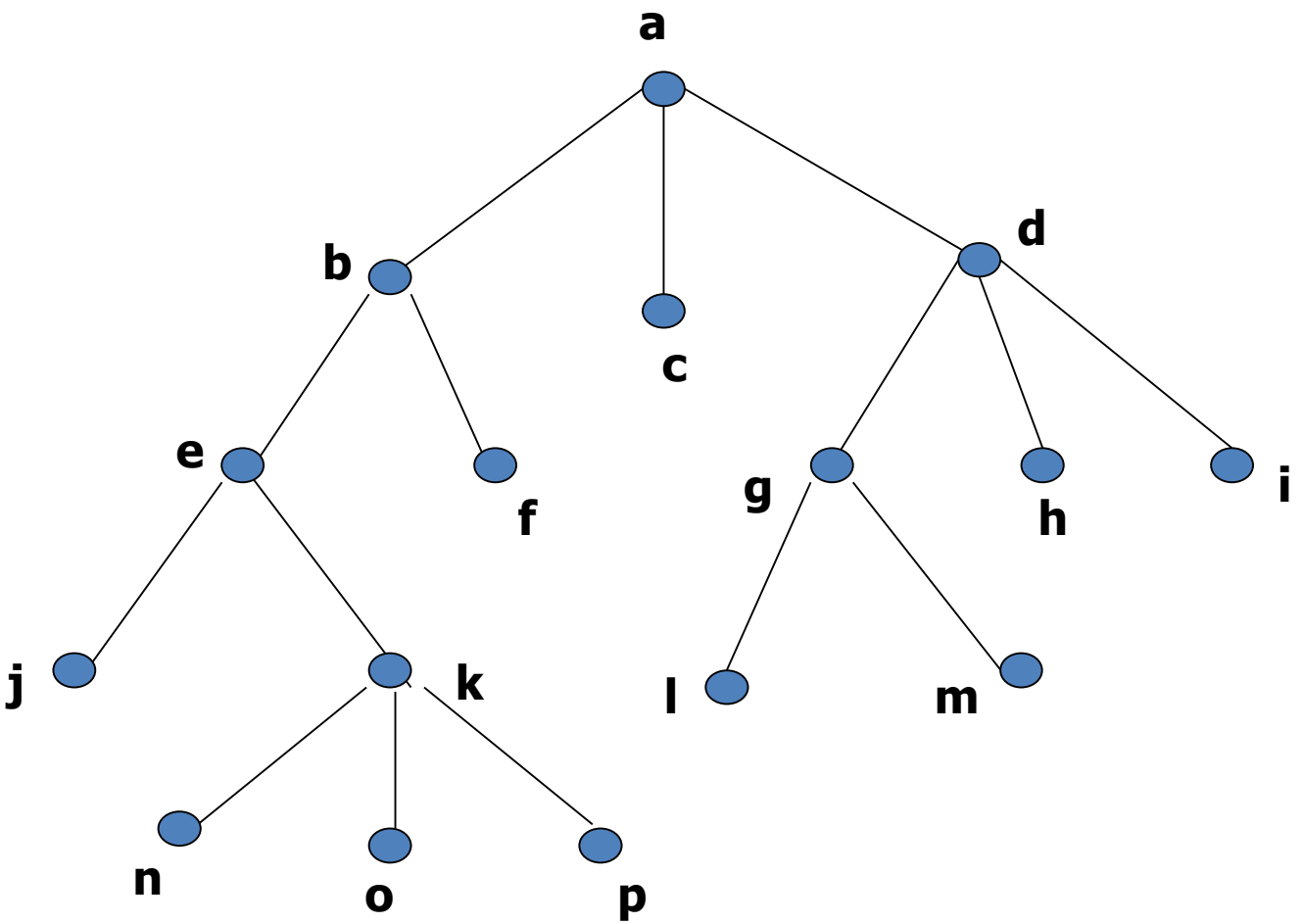
r = ab





Exercise:

Find the sequences for Preorder, Inorder and Postorder Traversal



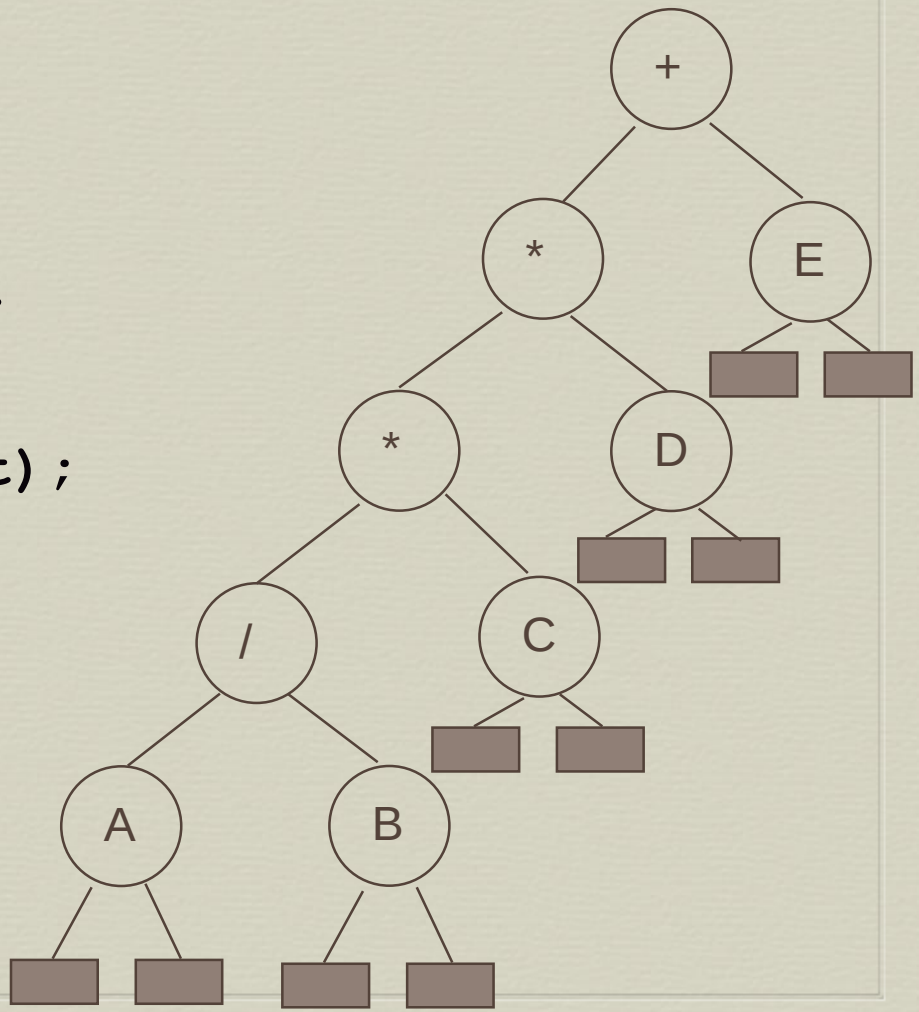
Answer:

- Preorder: **a b e j k n o p f c d g l m h i**
- Inorder: **j e n k o p b f a c l g m d h i**
- Postorder: **j n o p k e f b c l m g h i d a**

Inorder Traversal (recursive version)

A / B * C * D + E

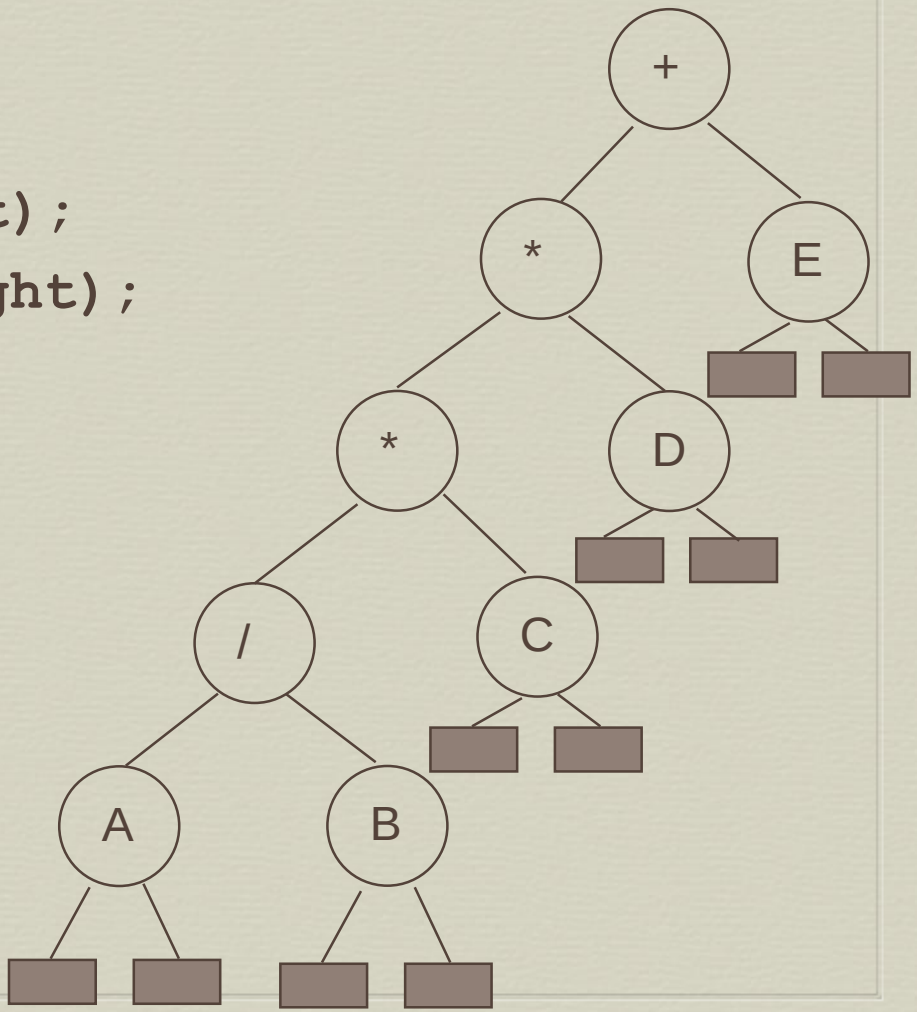
```
void inorder(TreeNode *ptr)
{
     if (ptr!=NULL) {
          inorder(ptr->left);
          cout << ptr->data;
          indorder(ptr->right);
     }
}
```



Postorder Traversal (recursive version)

```
void postorder(TreeNode *ptr)
{
    if (ptr!=NULL) {
        postorder(ptr->left);
        postdorder(ptr->right);
        cout << ptr->data;
    }
}
```

A B / C * D * E +



Preorder Traversal (recursive version)

```
void preorder(TreeNode *ptr)
{
    if (ptr!=NULL) {
        cout << ptr->data;
        preorder(ptr->left);
        predorder(ptr->right);
    }
}
```