



Topic 4

Intro to Trees

Lecture 4a

CSCI 240

Data Structures and Algorithms

Prof. Dominick Atanasio

Today

- This Class
 - Why different storage techniques?
 - Tree Terminology
 - Tree Traversals
 - Tree Representations

Algorithms & Storage

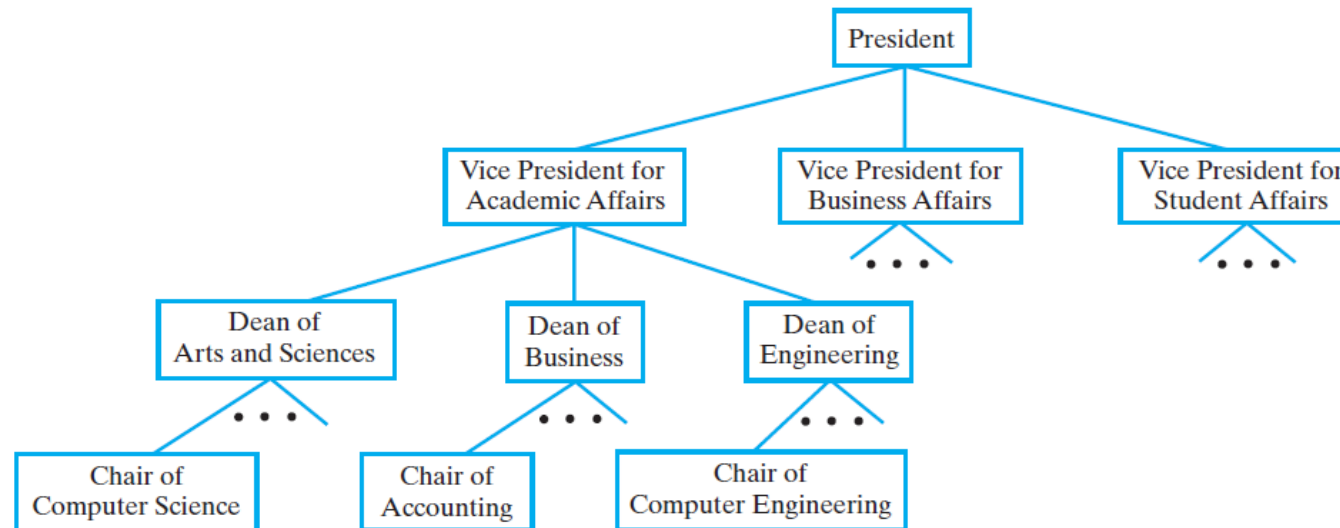
Storage	add item	Search item	Find min	Delete min
unsorted array	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Sorted	$O(n)$	$O(\log(n))$	$O(1)$	$O(n)$
Linked List	$O(1)$	$O(n)$	$O(n)$	$O(n)$
What if you need to optimize add or find and remove min				

Organization: Linear vs Nonlinear

- Linear organization
 - Such as Array, Linked List, Stack, Queue, and Dictionary
 - Objects appear one after the other.
- Nonlinear organization
 - Such as Tree and Graph
 - Objects are arranged hierarchically

Hierarchical Organizations

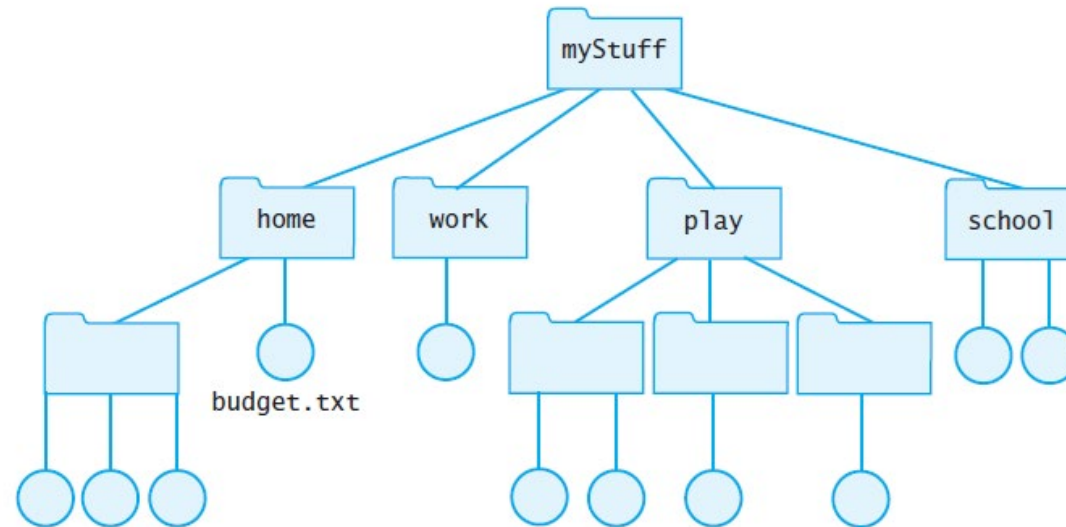
- (Example) A university's organization



A portion of a university's administrative structure

Hierarchical Organizations

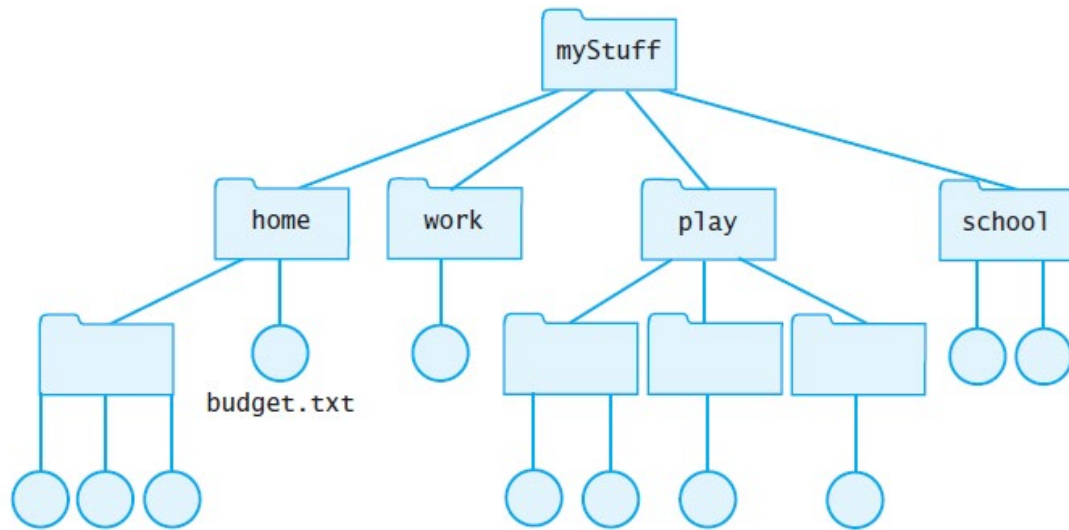
- (Example) File directories



Computer files organized into folders

Hierarchical Organizations

- (Example) File directories



Computer files organized into folders

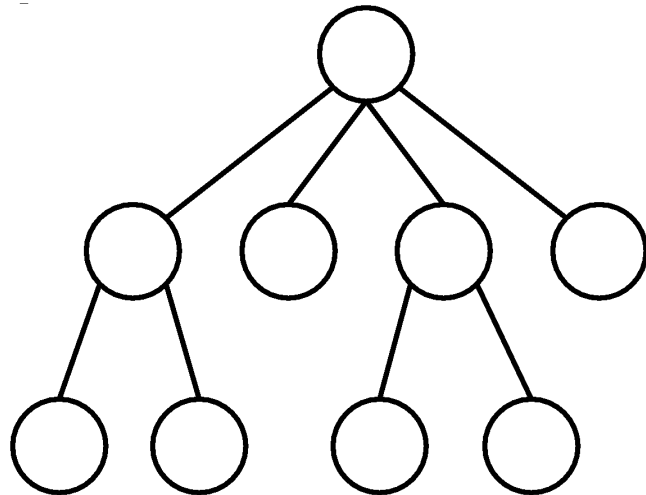
```
Select Command Prompt - tree
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\hji>tree
Folder PATH listing for volume OS
Volume serial number is 4223-D379
C:.
|-- .config
|   |-- octave
|   |   |-- 4.0.3
|   |   |-- qsci
|   |-- eclipse
|   |   |-- org.eclipse.epp.logging.aeri
|   |   |-- org.eclipse.oomph.p2
|   |   |   |-- cache
|   |   |-- org.eclipse.oomph.setup
|   |   |   |-- cache
|   |   |   |-- setups
|   |   |-- org.eclipse.recommenders.models.rcp
|   |   |-- repository
|   |   |   |-- http__download_eclipse_org_recommenders_models_neon_
|   |   |   |   |-- jre
|   |   |   |   |   |-- jre
|   |   |   |   |   |   |-- 1.0.0-SNAPSHOT
|   |   |   |   |-- org
|   |   |   |   |   |-- eclipse
|   |   |   |   |   |   |-- recommenders
|   |   |   |   |   |   |   |-- index
|   |   |   |   |   |   |   |   |-- 0.0.0-SNAPSHOT
|   |-- .ipython
```

Tree command

Trees Definition

- In computer science, a tree is a widely-used data organization to place data in hierarchical structure.
- Tree
 - A collection of nodes connected by edges without having any cycle.



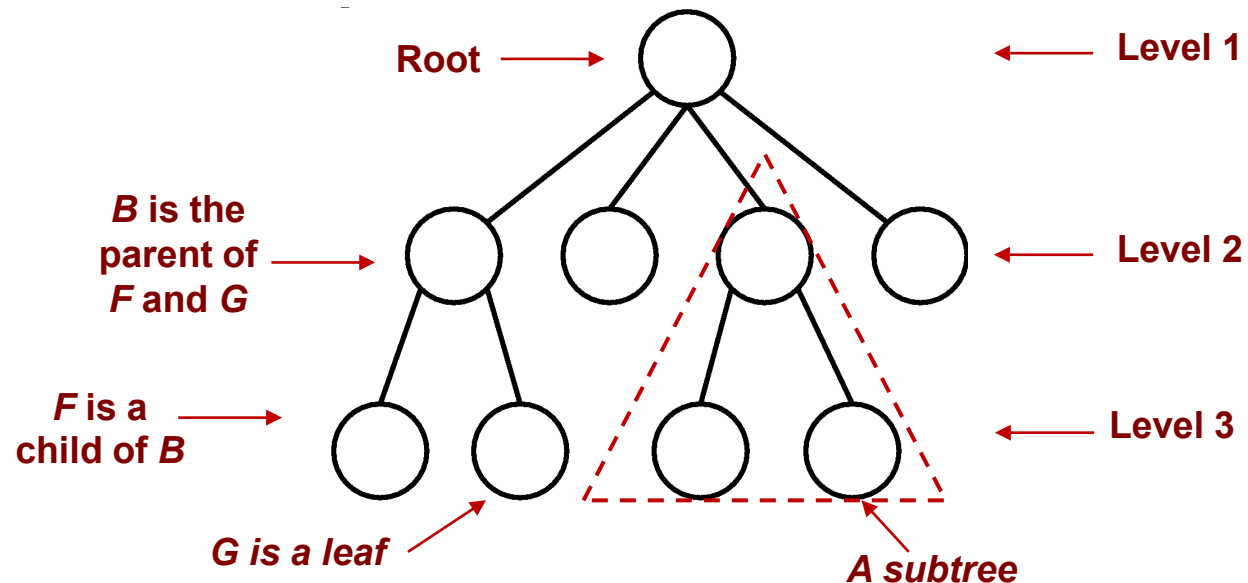
Tree (Data Structure)



Real Tree

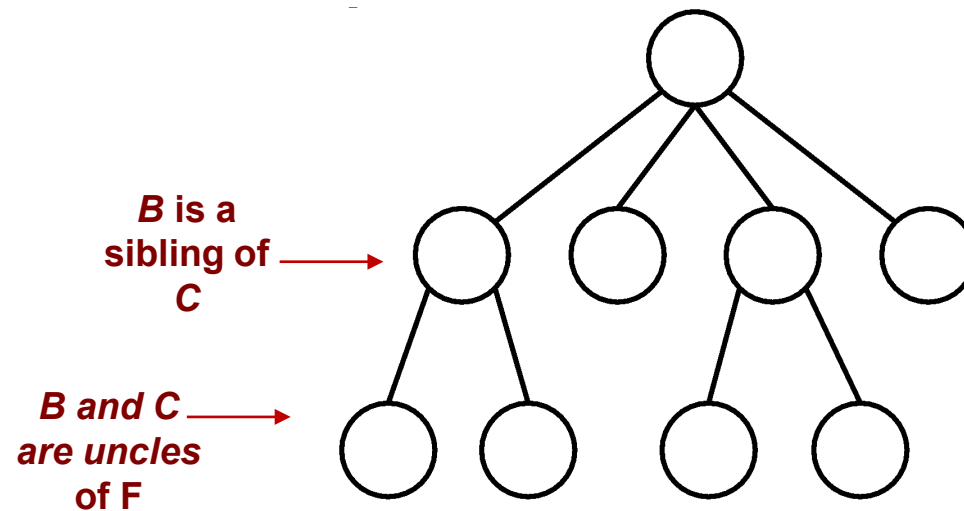
Tree Terminology

- Terminology
 - Level: the level of a node represents that node's hierarchy
 - Root: a single node at the top level
 - Children: the nodes at each successive level of a tree are the children of the nodes at the previous level.
 - Parent: a node that has children is the parent of those children
 - Leaf: a node has no children
 - Subtree: any node and its descendants form a subtree of the original tree



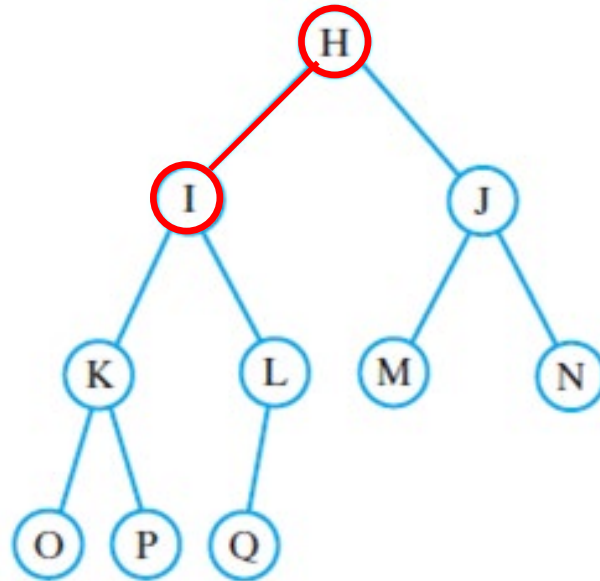
Tree Terminology

- Terminology
 - Sibling: Sibling nodes share the same parent node
 - Uncles: Siblings of that node's parent.
 - Ancestor: A node that is connected to all lower-level nodes.
 - Descendant: The connected lower-level nodes are "descendants" of the ancestor node.



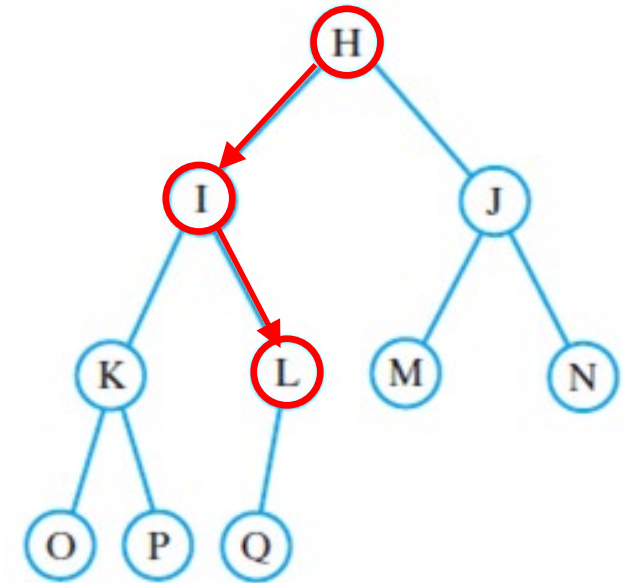
Tree Terminology

- Edge: connection between one node to another



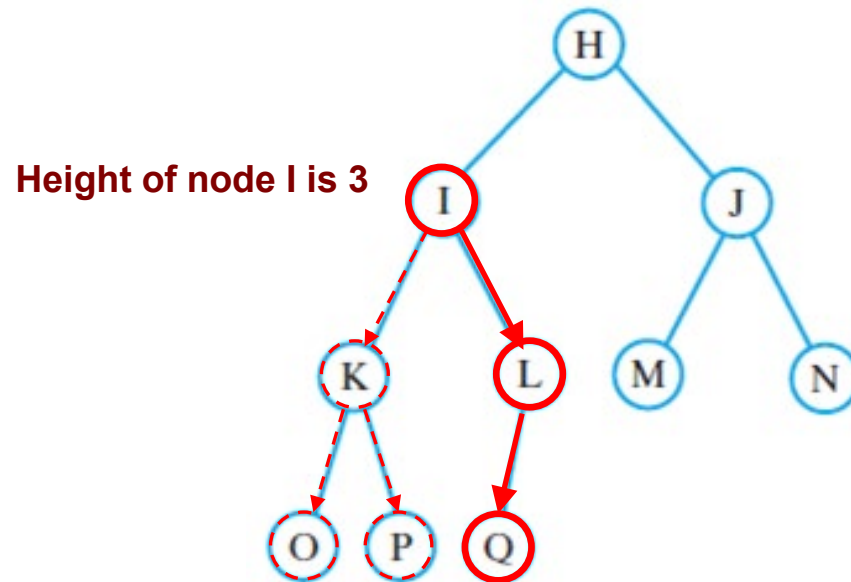
Tree Terminology

- Edge: connection from one node to another
- Path: A sequence of nodes and edges connecting a node with a descendant.



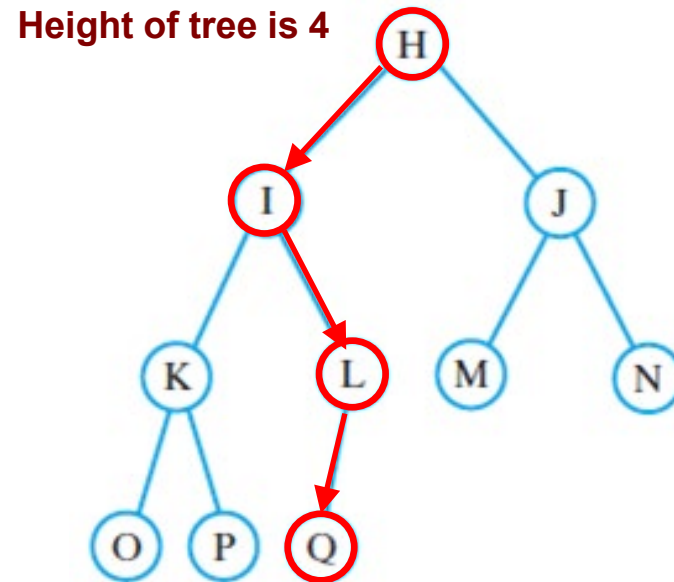
Tree Terminology

- Edge: connection between one node to another
- Path: A sequence of nodes and edges connecting a node with a descendant.
- Height of node: The height of a node is the number of edges on the longest path between that node and a leaf + 1 (or the number of nodes on the longest path between that node and a leaf).



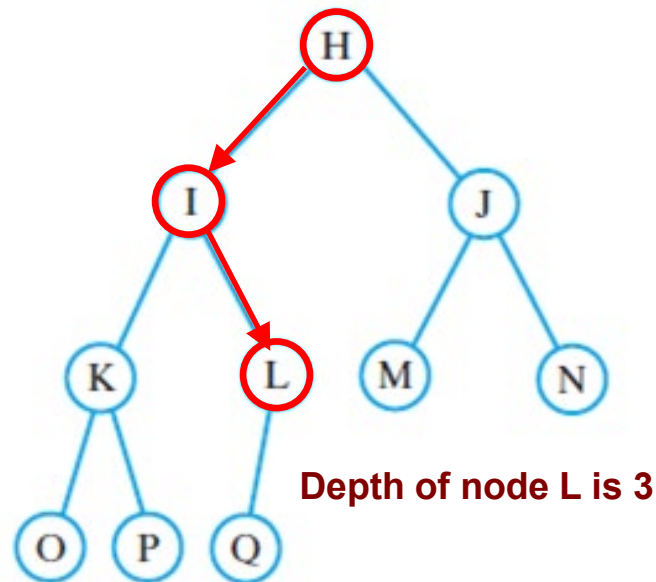
Tree Terminology

- Edge: connection between one node to another
- Path: A sequence of nodes and edges connecting a node with a descendant.
- Height of node: The height of a node is the number of edges on the longest path between that node and a leaf + 1.
- Height of tree: The height of a tree is the height of its root node.



Tree Terminology

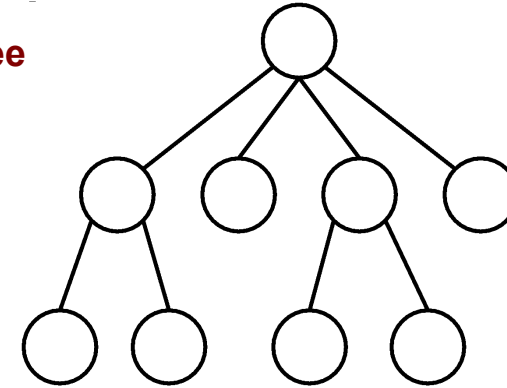
- Edge: connection between one node to another
- Path: A sequence of nodes and edges connecting a node with a descendant.
- Height of node: The height of a node is the number of edges on the longest path between that node and a leaf + 1.
- Height of tree: The height of a tree is the height of its root node.
- Depth of node: The depth of a node is the number of edges from the tree's root node to the node + 1.



Tree Terminology

- In general, a tree can have an arbitrary number of children.
- **N-ary Tree:** each node has no more than n children.

4-ary tree



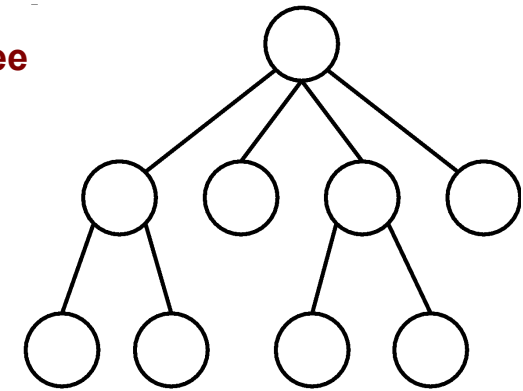
Tree Terminology

- In general, a tree can have an arbitrary number of children.

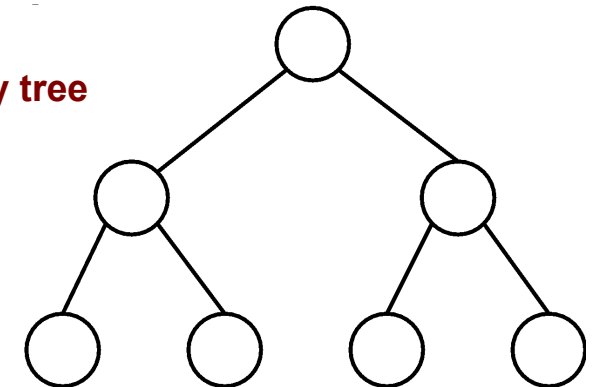
- **N-ary Tree:** each node has no more than n children.
- **Binary Tree:** each node has at most two children.

Ternary Tree: each node has at most 3 children

4-ary tree

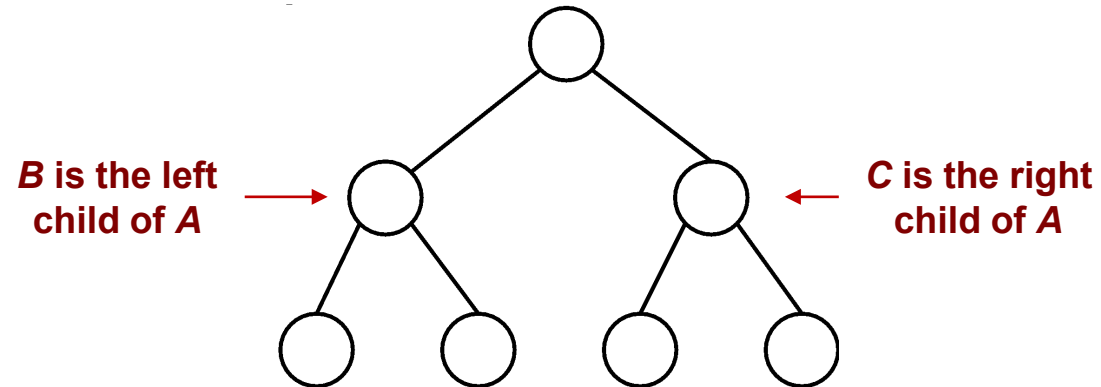


Binary tree



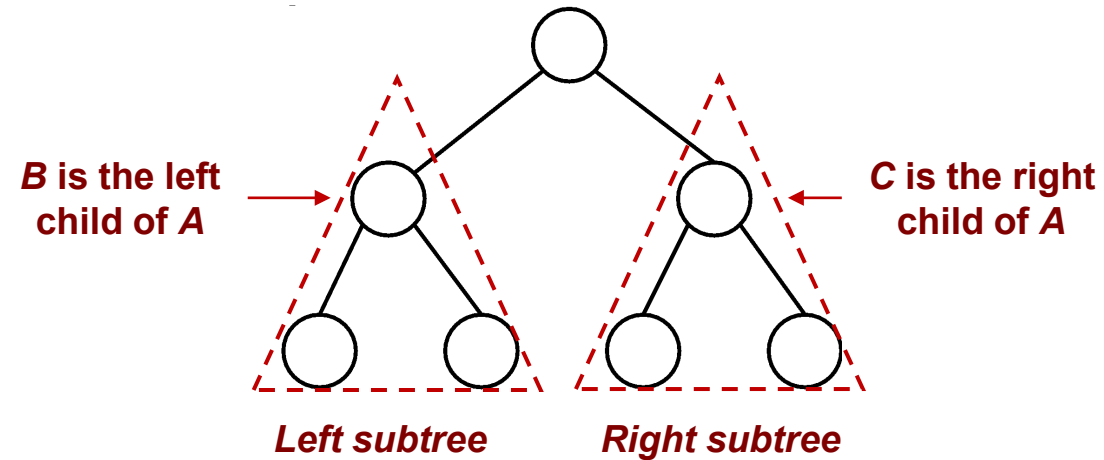
Tree Terminology

- Binary Tree
 - Every node in a tree can have at most two children.



Tree Terminology

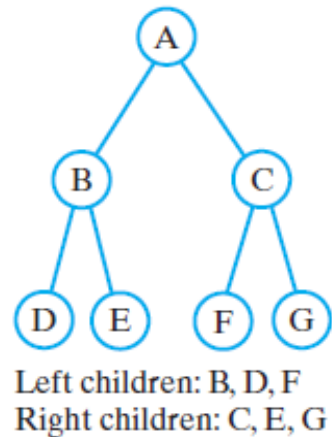
- Binary Tree
 - Every node in a tree can have at most two children.



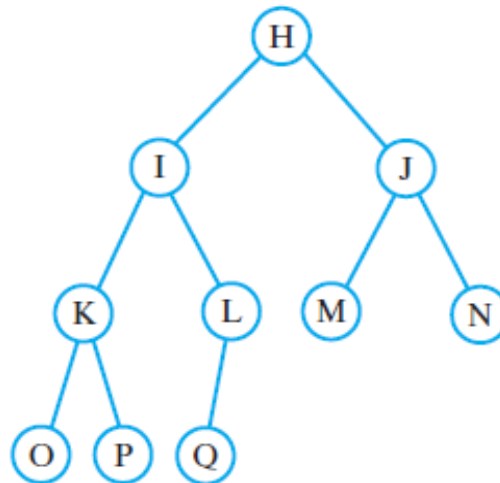
Tree Terminology

- Full Binary Tree
 - All internal nodes have two children, and all leaves are at the same depth.
- Complete Binary Tree
 - An almost-full binary tree; the bottom level of the tree is filling from left to right but may not have its full complement of leaves.

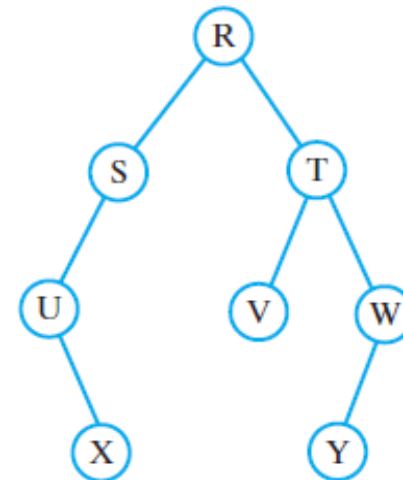
(a) Full tree



(b) Complete tree

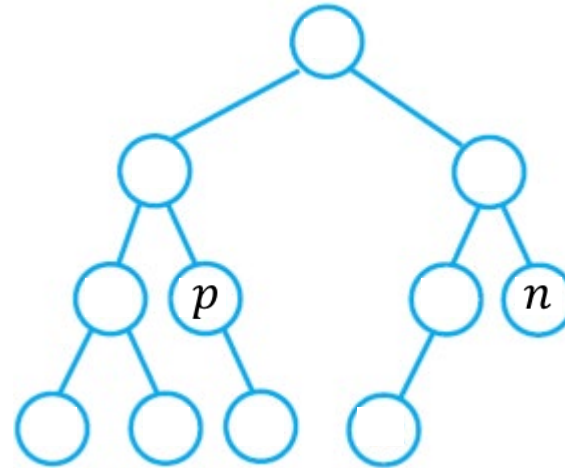


(c) Tree that is not full and not complete



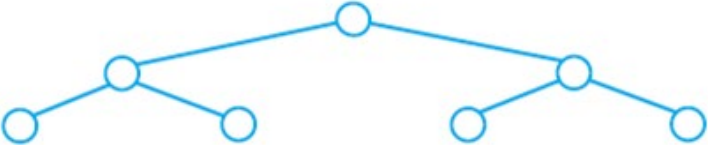


In-Class Exercise

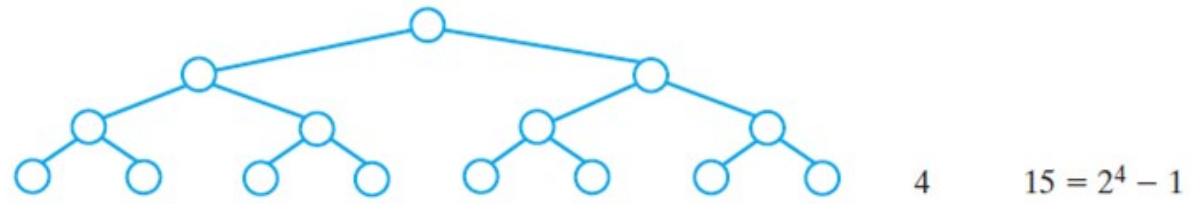
- What is the height of the tree?
- What is the height of node n ?
- What is the height of node p ?
- What is the depth of node n ?
- What is the depth of node p ?
- Is the tree complete?



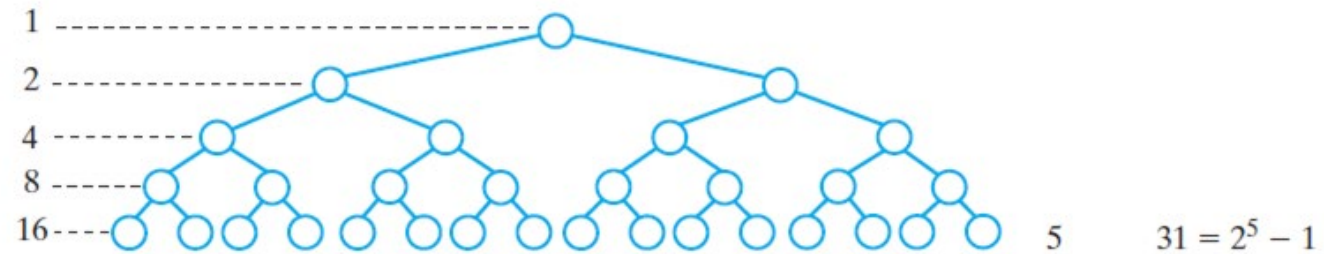
Height of Full or Complete Binary Trees

Full Tree	Height	Number of Nodes
	1	$1 = 2^1 - 1$
	2	$3 = 2^2 - 1$
	3	$7 = 2^3 - 1$

Height of Full or Complete Binary Trees



Number of
nodes per level



Height of Full or Complete Binary Trees

Full Tree

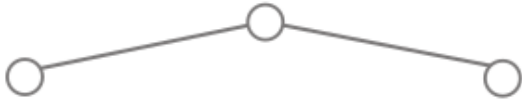
Height

Number
of Nodes



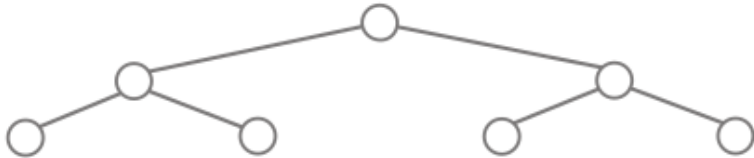
1

$$1 = 2^1 - 1$$



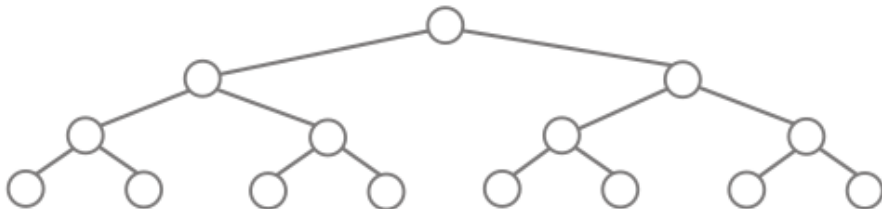
2

$$3 = 2^2 - 1$$



3

$$7 = 2^3 - 1$$



4

$$15 = 2^4 - 1$$

The number of nodes in a full binary tree of height h is: $n = 2^h - 1$

The height of a full or complete binary tree having n nodes: $h = \text{ceil}(\log_2(n+1))$

The number of leaf nodes in a full binary tree having a height h is $n = 2^{h-1}$

The number of non-leaf nodes in a full binary tree having a height h is $n = 2^{h-1} - 1$

The number of leaf nodes in a complete binary tree having a number of nodes n :

$$l = \text{floor}((n+1)/2)$$

The number of non-leaf nodes in a complete binary tree having a number of nodes n : $nl = \text{ceil}((n-1)/2)$

In-Class Exercise

- In a full binary tree of height 4,
 - How many nodes are in the tree?
 - How many leaves are in the tree?

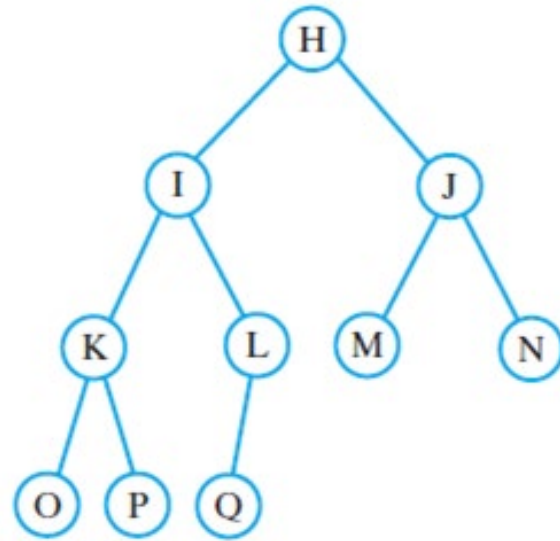
- In a complete binary tree with 10 nodes,
 - What is the height of the tree?
 - How many non-leaf nodes are in the tree?
 - How many leaves are in the tree?

In-Class Exercise

- In a complete binary tree of height h ,
 - How many nodes are in the tree?
 - How many leaves are in the tree?
- In a complete binary tree with n nodes,
 - What is the height of the tree?
 - How many non-leaf nodes are in the tree?
 - How many leaves are in the tree?

Tree Terminology

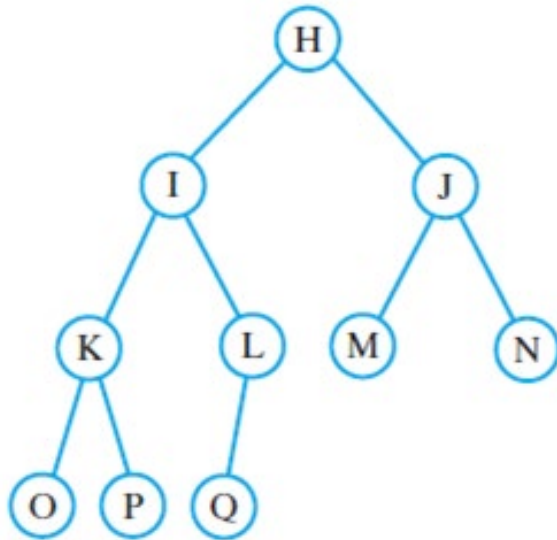
- **Balanced Binary Trees**
 - The subtrees of each node in the tree differ in height by no more than 1.



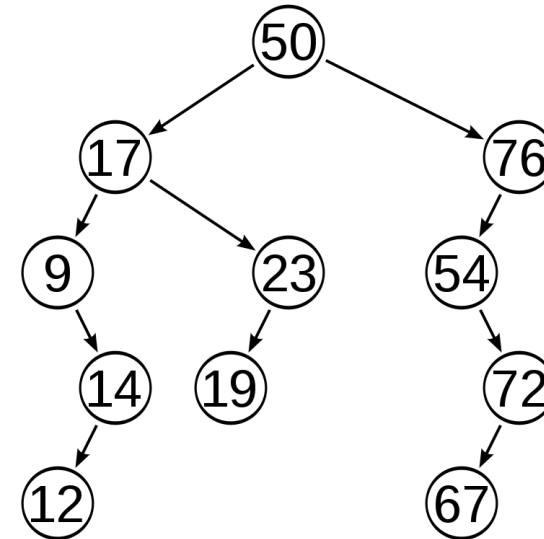
A balanced binary tree

Tree Terminology

- **Balanced Binary Trees**
 - The subtrees of each node in the tree differ in height by no more than 1.



A balanced binary tree



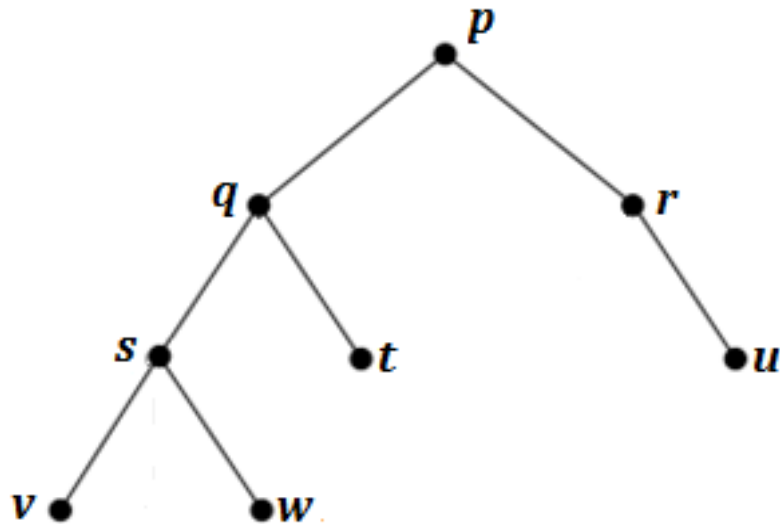
Is this a balanced binary tree?

Tree Traversals

- If a tree structure is being used to store data, it is often helpful to have a systematic mechanism for writing out the data values stored at all the nodes.
- This can be accomplished by traversing the tree
 - Visiting each of the nodes in the tree structure.
- The common tree traversals are
 - Preorder traversal
 - Inorder traversal
 - Postorder traversal
 - Level-order traversal

Tree Traversals

- Pre-order traversal:
 - Process the root.
 - Process the nodes in the left subtree with a recursive call.
 - Process the nodes in the right subtree with a recursive call.

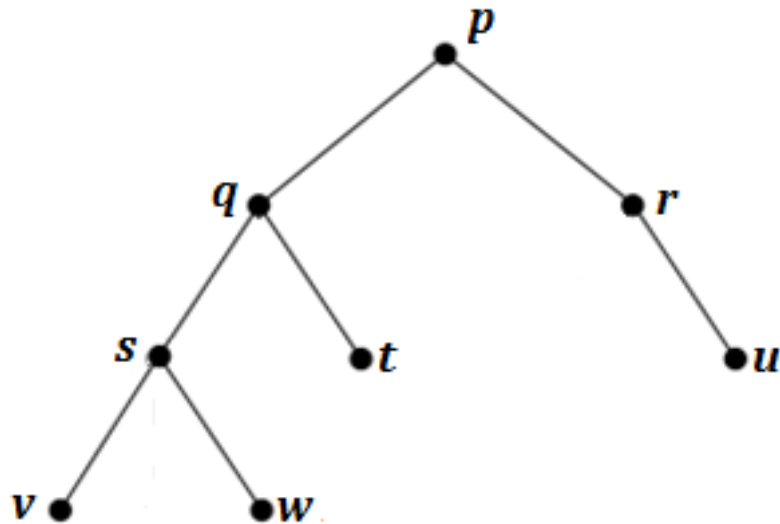


The preorder (root, left, right) traversal produces:

p, q, s, v, w, t, r, u

Tree Traversals

- In-order traversal:
 - Process the nodes in the left subtree with a recursive call.
 - Process the root.
 - Process the nodes in the right subtree with a recursive call.

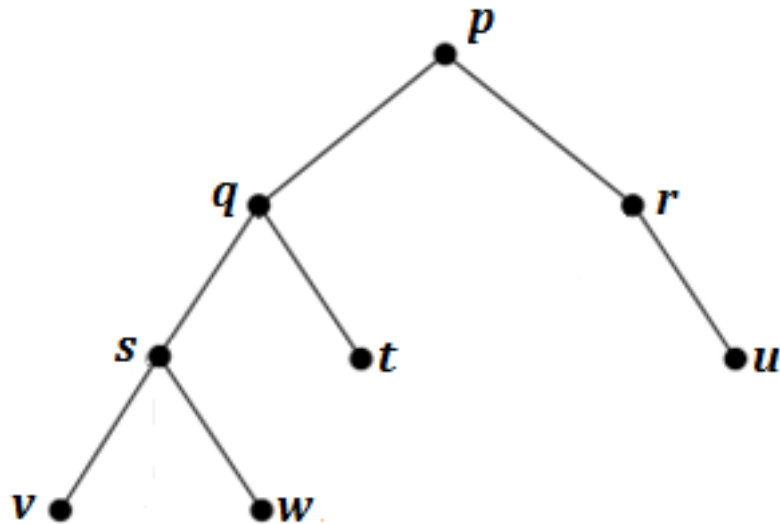


The inorder (left, root, right) traversal produces:

v, s, w, q, t, p, r, u

Tree Traversals

- Post-order traversal:
 - Process the nodes in the left subtree with a recursive call.
 - Process the nodes in the right subtree with a recursive call.
 - Process the root.

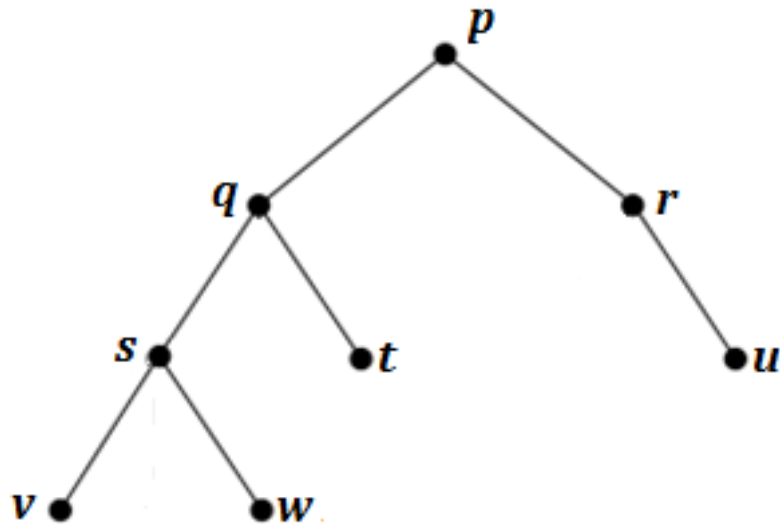


The postorder (left, right, root) traversal produces:

v, w, s, t, q, u, r, p

Tree Traversals

- Level-order traversal:
 - Process the root.
 - Process nodes one level at a time (visiting nodes in each level from left to right)

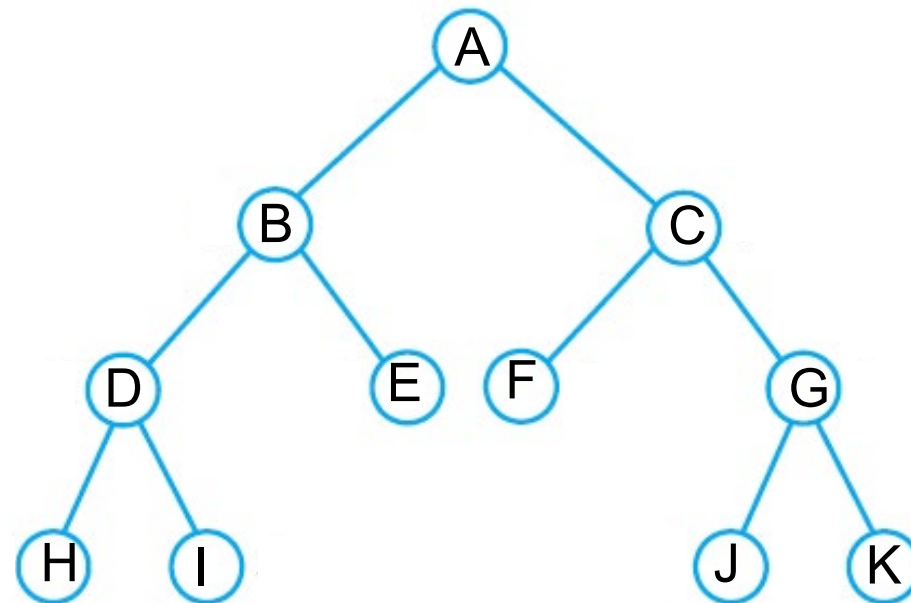


The level-order traversal produces:

p, q, r, s, t, u, v, w

In-Class Exercise

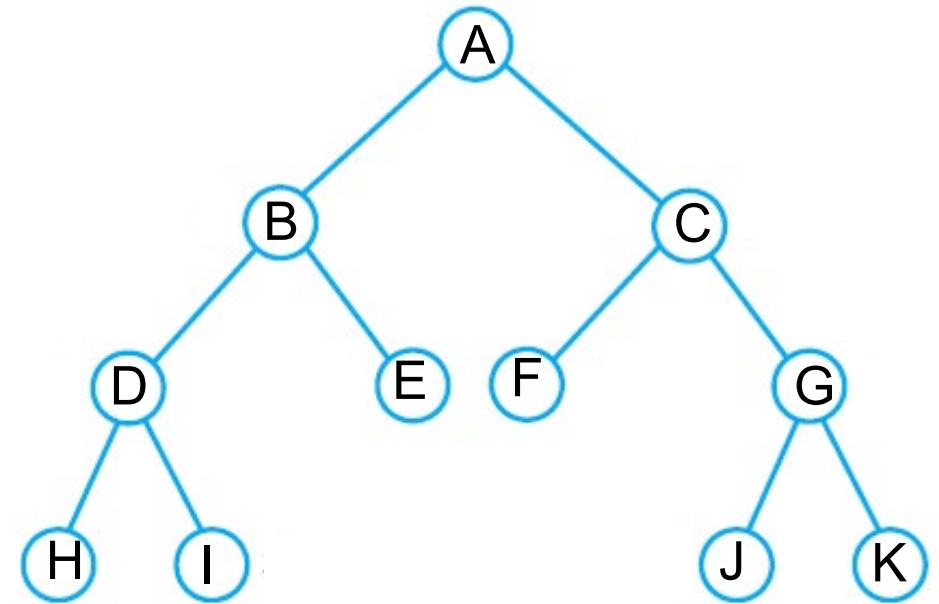
- Find the pre-order, in-order, post-order, and level-order traversal for the following tree:



In-Class Exercise

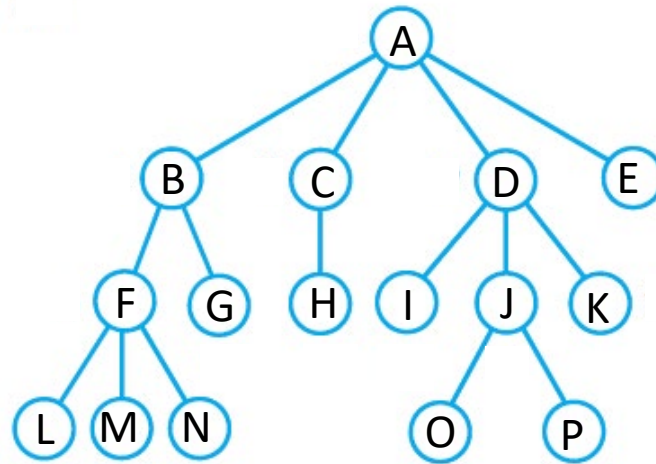
- Find the pre-order, in-order, post-order, and level-order traversal for the following tree:

Pre-order: *A, B, D, H, I, E, C, F, G, J, K*
In-order: *H, D, I, B, E, A, F, C, J, G, K*
Post-order: *H, I, D, E, B, F, J, K, G, C, A*
Level-order: *A, B, C, D, E, F, G, H, I, J, K*



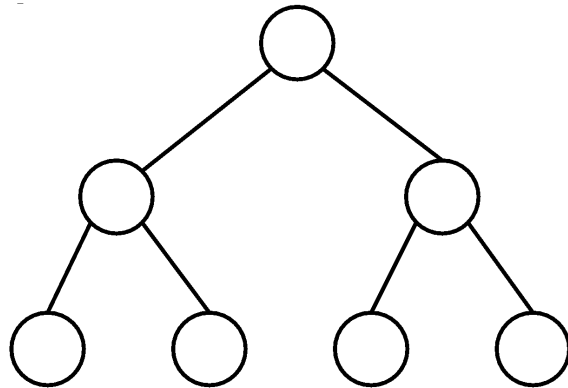
In-Class Exercise

- Find the pre-order, in-order, post-order, and level-order traversal for the following tree:

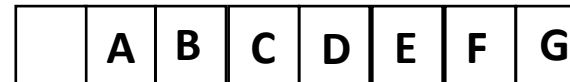


Tree Representations

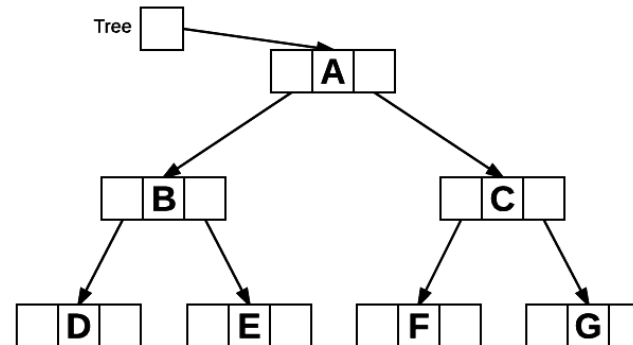
- Storage choices:
 - Array representation (contiguous structure)
 - Node representation (linked list structure)



A binary tree



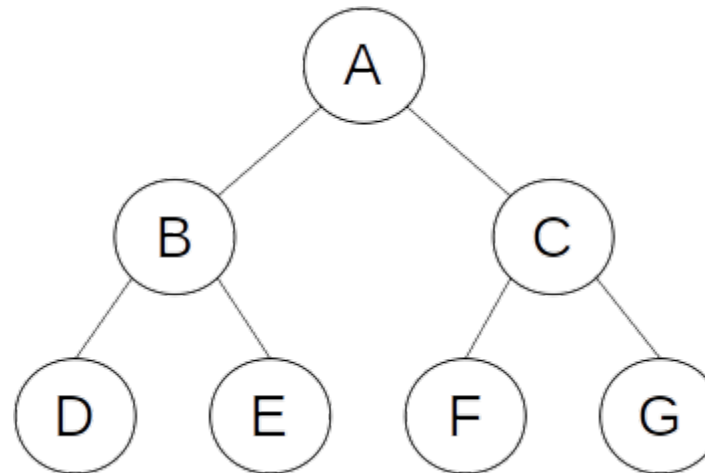
Array representation



Node representation

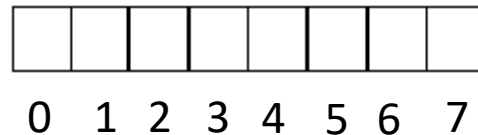
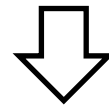
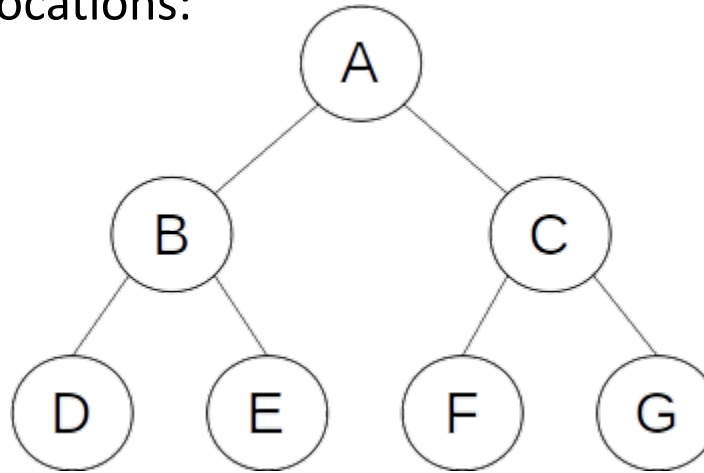
Array Representation

- Formulas for the array representation:
 - The data from the root always appears in the [1] element of the array.
 - Suppose the data for a node appears in component [i] of the array. Then its children (if they exist) always have their data at these locations:
 - Left child at component $[2i]$
 - Right child at component $[2i + 1]$



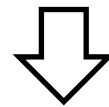
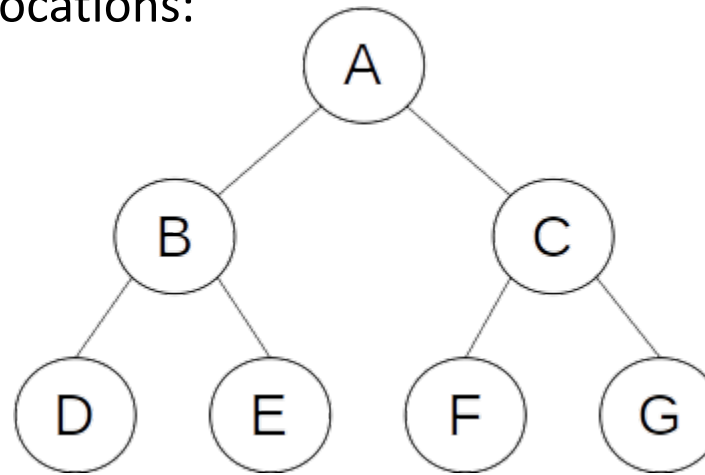
Array Representation

- Formulas for the array representation:
 - The data from the root always appears in the [1] component of the array.
 - Suppose the data for a node appears in component [i] of the array. Then its children (if they exist) always have their data at these locations:
 - Left child at component $[2i]$
 - Right child at component $[2i + 1]$



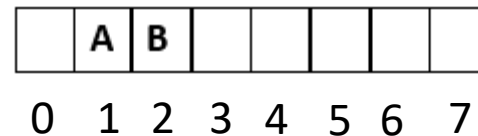
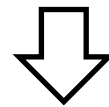
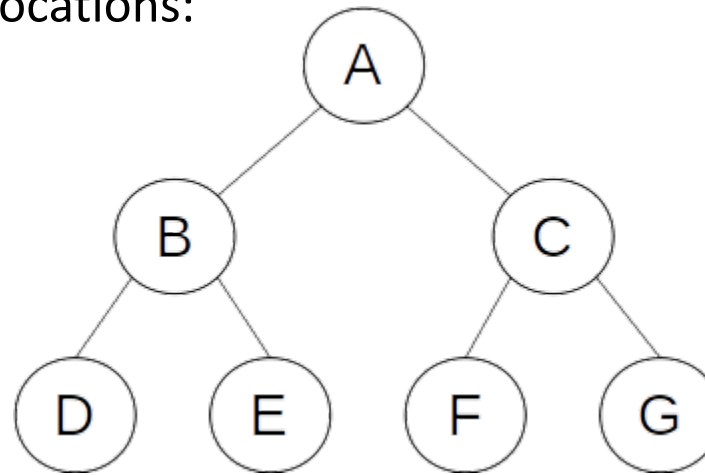
Array Representation

- Formulas for the array representation:
 - The data from the root always appears in the [1] component of the array.
 - Suppose the data for a node appears in component [i] of the array. Then its children (if they exist) always have their data at these locations:
 - Left child at component $[2i]$
 - Right child at component $[2i + 1]$



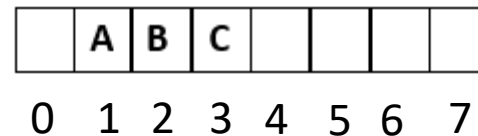
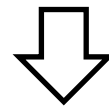
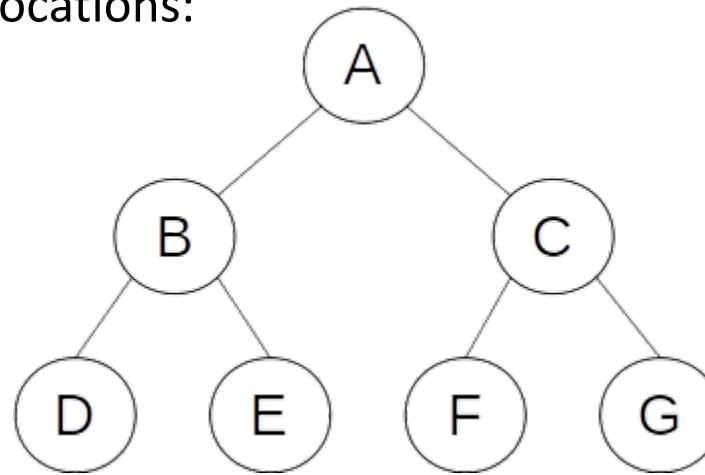
Array Representation

- Formulas for the array representation:
 - The data from the root always appears in the [1] component of the array.
 - Suppose the data for a node appears in component [i] of the array. Then its children (if they exist) always have their data at these locations:
 - Left child at component $[2i]$
 - Right child at component $[2i + 1]$



Array Representation

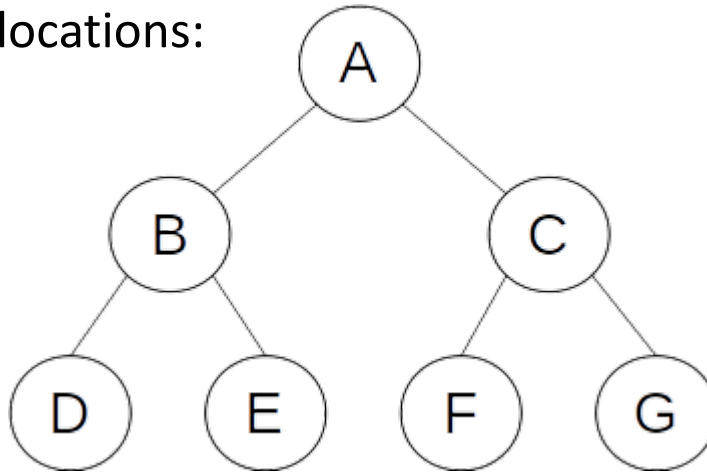
- Formulas for the array representation:
 - The data from the root always appears in the [1] component of the array.
 - Suppose the data for a node appears in component [i] of the array. Then its children (if they exist) always have their data at these locations:
 - Left child at component $[2i]$
 - Right child at component $[2i + 1]$



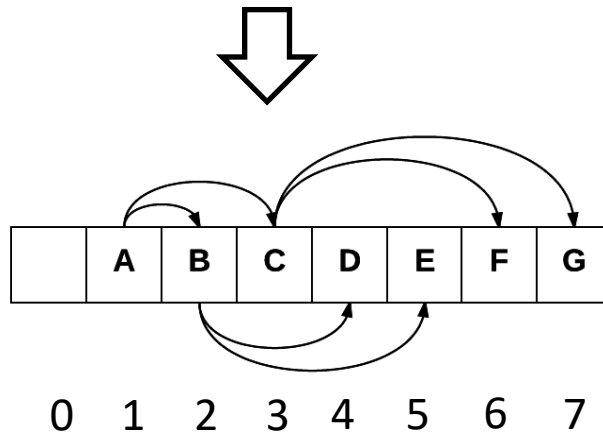
Array Representation

- Formulas for the array representation:

- The data from the root always appears in the [1] component of the array.
- Suppose the data for a node appears in component [i] of the array. Then its children (if they exist) always have their data at these locations:
 - Left child at component $[2i]$
 - Right child at component $[2i + 1]$



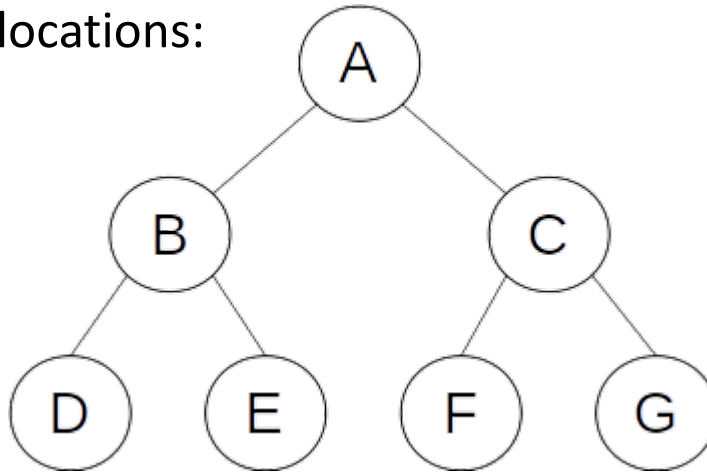
Question: Suppose the data for a non-root node appears in component [i] of the array. What is the location of the data for its parent?



The actual links between the nodes are not stored but are determined by the formula.

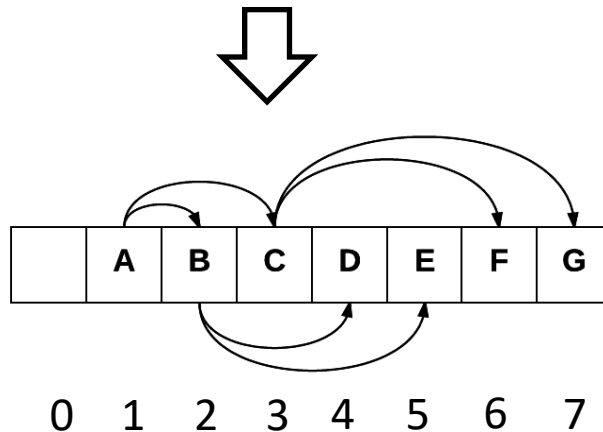
Array Representation

- Formulas for the array representation:
 - The data from the root always appears in the [1] component of the array.
 - Suppose the data for a node appears in component [i] of the array. Then its children (if they exist) always have their data at these locations:
 - Left child at component $[2i]$
 - Right child at component $[2i + 1]$



Question: Suppose the data for a non-root node appears in component [i] of the array. What is the location of the data for its parent?

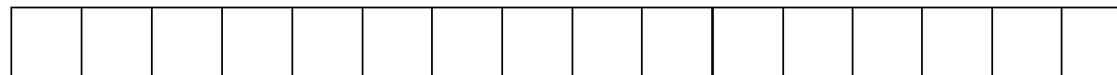
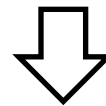
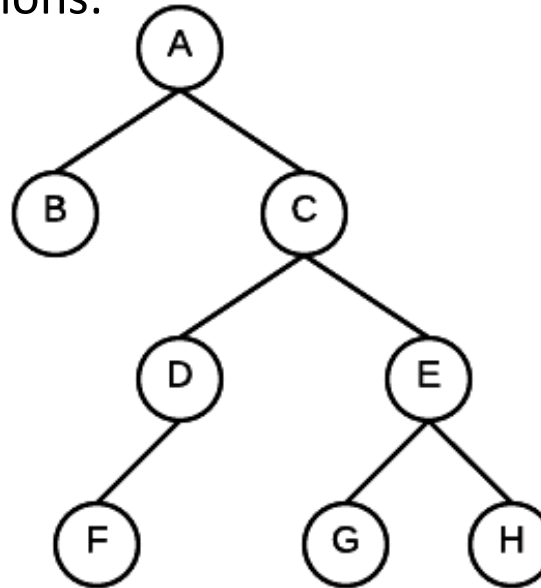
At array index $i/2$



The actual links between the nodes are not stored but are determined by the formula.

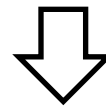
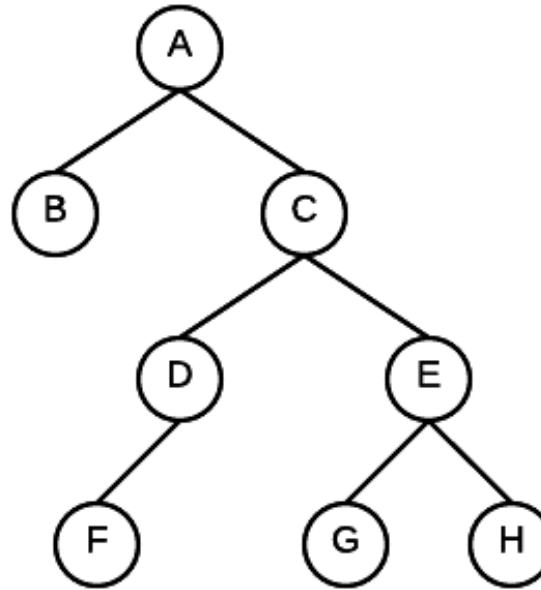
In-Class Exercise

- Formulas for the array representation:
 - The data from the root always appears in the [1] component of the array.
 - Suppose the data for a node appears in component [i] of the array. Then its children (if they exist) always have their data at these locations:
 - Left child at component $[2i]$
 - Right child at component $[2i + 1]$



In-Class Exercise

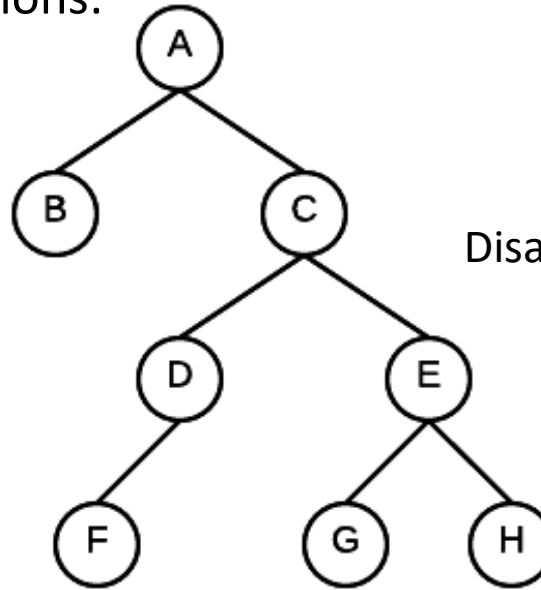
- Formulas for the array representation:
 - The data from the root always appears in the [1] component of the array.
 - Suppose the data for a node appears in component [i] of the array. Then its children (if they exist) always have their data at these locations:
 - Left child at component $[2i]$
 - Right child at component $[2i + 1]$



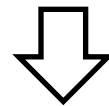
	A	B	C			D	E					F		G	H
--	---	---	---	--	--	---	---	--	--	--	--	---	--	---	---

In-Class Exercise

- Formulas for the array representation:
 - The data from the root always appears in the [1] component of the array.
 - Suppose the data for a node appears in component [i] of the array. Then its children (if they exist) always have their data at these locations:
 - Left child at component $[2i]$
 - Right child at component $[2i + 1]$



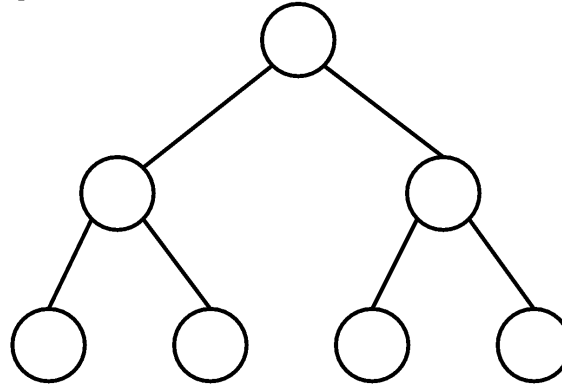
Disadvantages of an array implementation of a tree:
The memory wasted in an unbalanced tree.
Also, a fixed array size makes it harder to grow.



	A	B	C			D	E					F		G	H
--	---	---	---	--	--	---	---	--	--	--	--	---	--	---	---

Node Representation

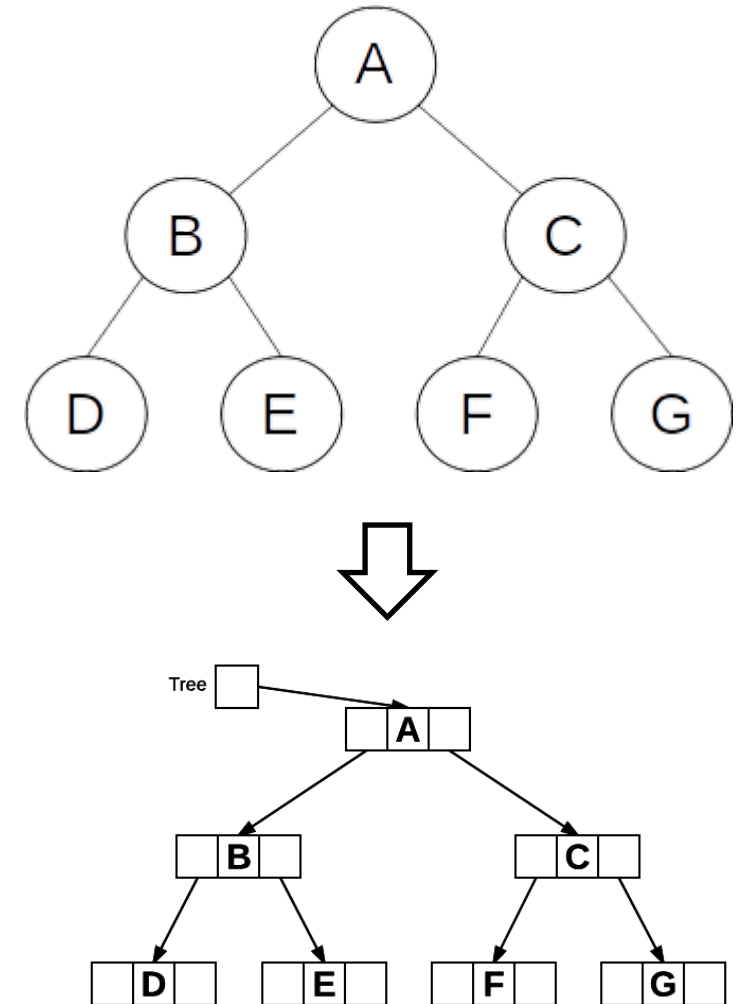
- Each node of a binary tree can be stored as an object of a binary tree node class.
- The class contains private instance variables that are pointers to other nodes in the tree.
- An entire tree is represented as a pointer to the root node.



Node Representation

- Each node of a binary tree can be stored as an object of a binary tree node class.
- The class contains instance variables that are pointers to other nodes in the tree.
- An entire tree is represented as a pointer to the root node.

```
template<typename T>
struct Node
{
    T item;
    Node<T> leftChild;
    Node<T> rightChild;
    ...
};
```



Node Representation

- Each node of a binary tree can be stored as an object of a binary tree node class.
- The class contains instance variables that are pointers to other nodes in the tree.
- An entire tree is represented as a pointer to the root node.
 - The pointer to the root is similar to the head of a linked list, providing a starting point to access all the nodes in the tree.
 - We could include other things in the tree node.

```
template<typename T>
struct Node
{
    T item;
    Node<T> leftChild;
    Node<T> rightChild;
    ...
};
```

