CMSC 341
Lecture 9
Introduction to Trees

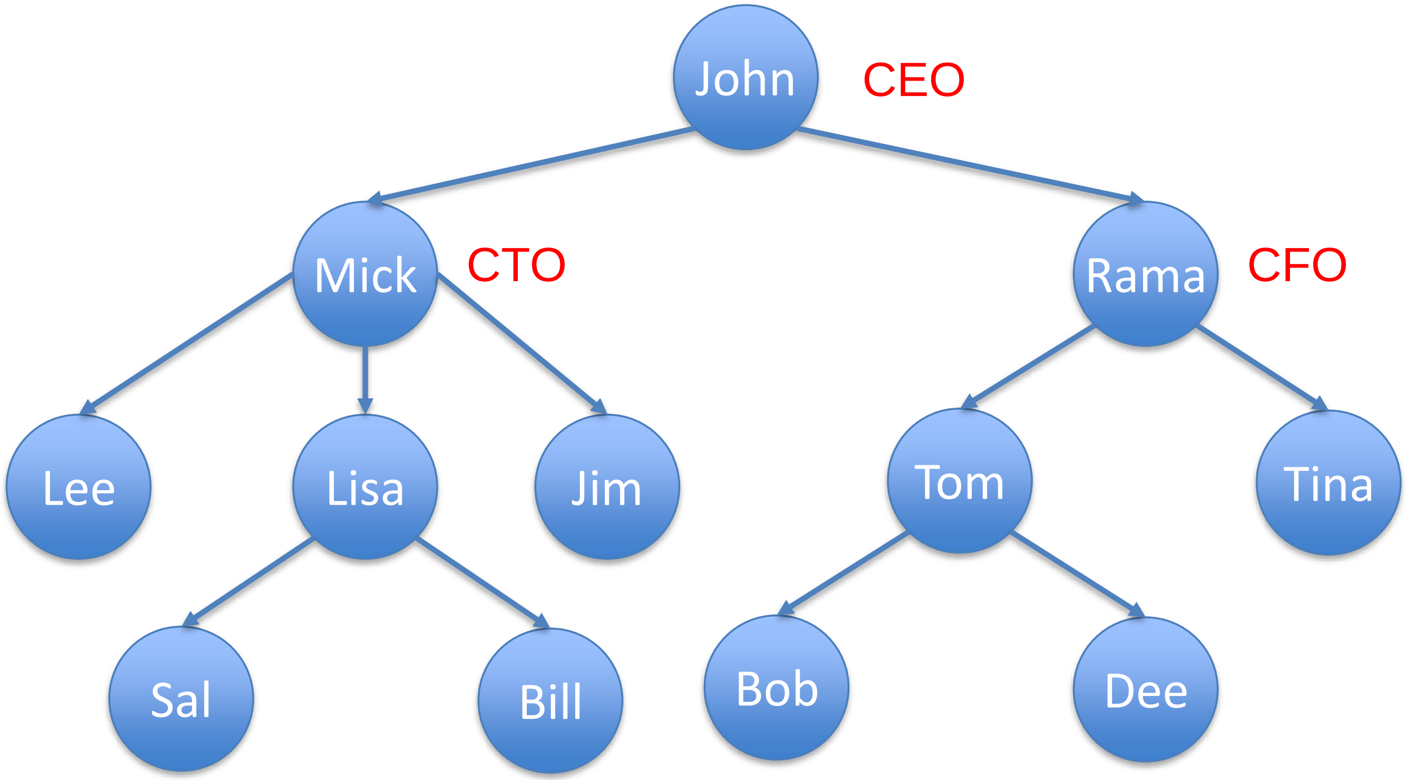
(Adapted from Profs. Gibson & Dixon's slides)

Introduction to Trees

What is a Tree?

- In computer science, a *tree* is an abstract model of a hierarchical structure
- Applications:
  - Organization charts
  - File systems
  - Programming environments

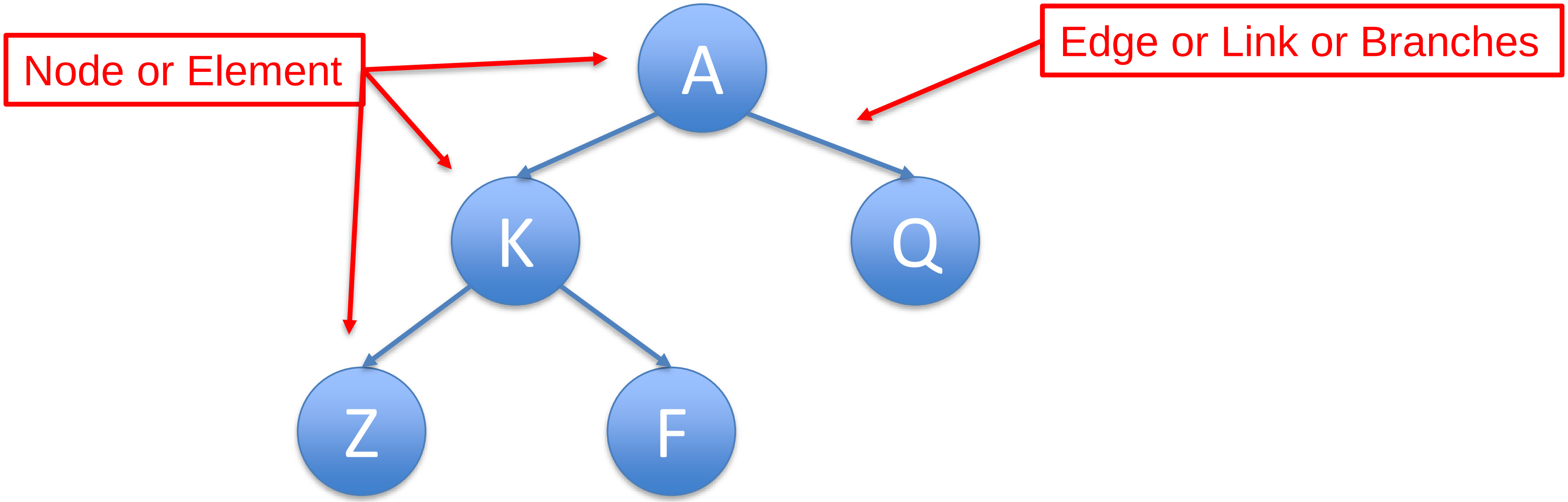
Tree Example – Org Chart



What is a Tree?

- A tree is a special form of a graph; it consists of:
  - Elements, called *nodes* or *vertices*
  - Connections, called *edges* or *arcs*
- A tree has the following additional properties:
  - The edges are directional (have arrows)
  - All nodes but one (the root node) have exactly one edge coming in (i.e., pointing to it) and 0 or more going out
  - There are no *cycles* (loops)

What is a Tree?

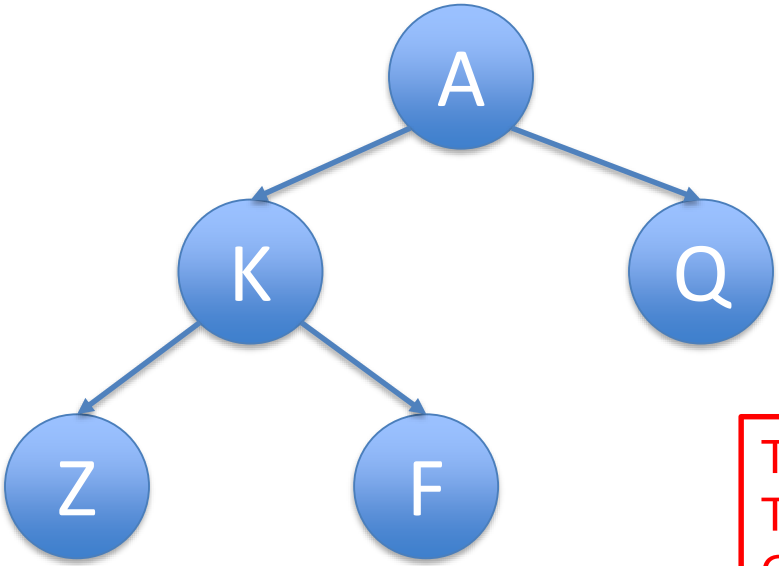


Tree Terminology

- There are two main ways that trees are described.
  1. Terms are related to “trees” such as *root*, *branches*, and *leaves*
  2. Terms are related to “ancestry” such as *parent*, *children*, *sibling*, *ancestors*, and *descendants*

What is a Tree?

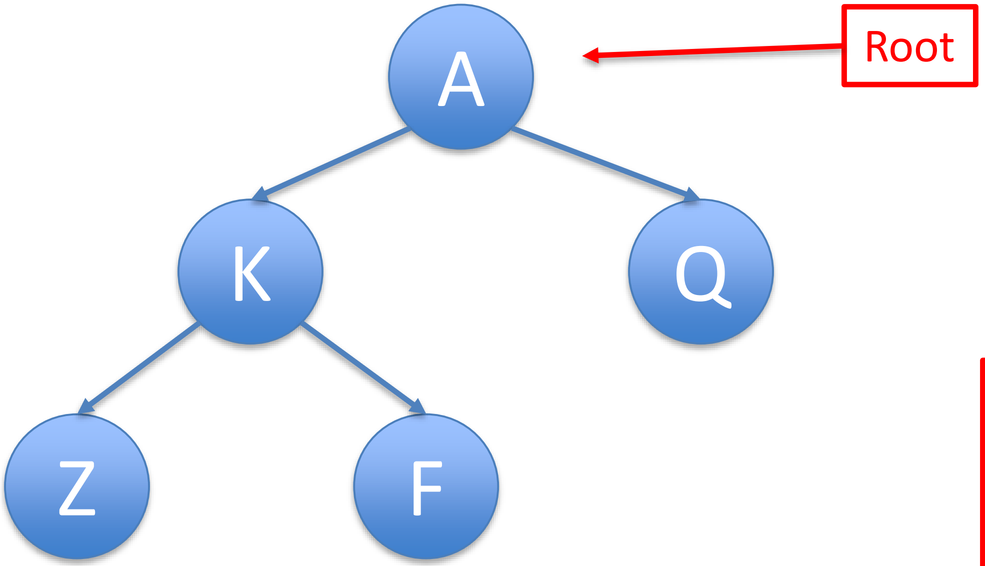
- Each node may have 0 or more ***children***



The children of A are K and Q.
The children of K are Z and F.
Q, Z, and F have no children.

What is a Tree?

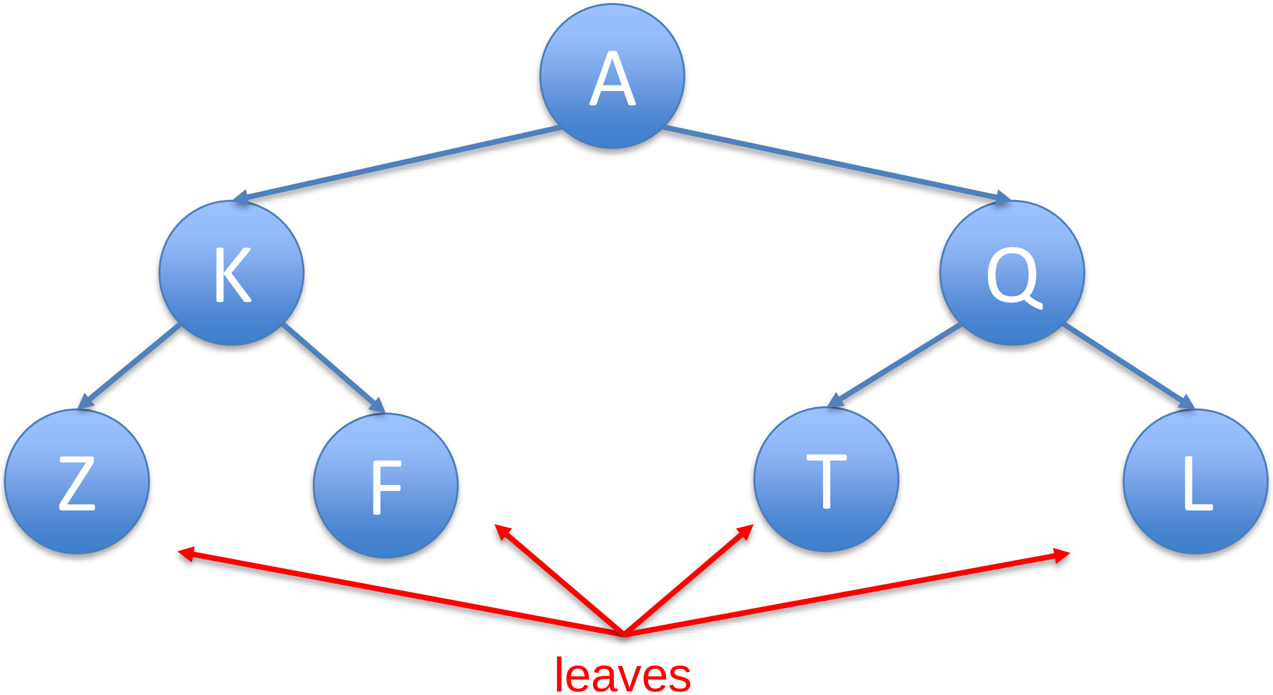
- Each node has **exactly one** *parent*
  - Except the starting / top node, called the *root*



The parent of K is A.
The parent of Q is A.
The parent of Z is K.
The parent of F is K.

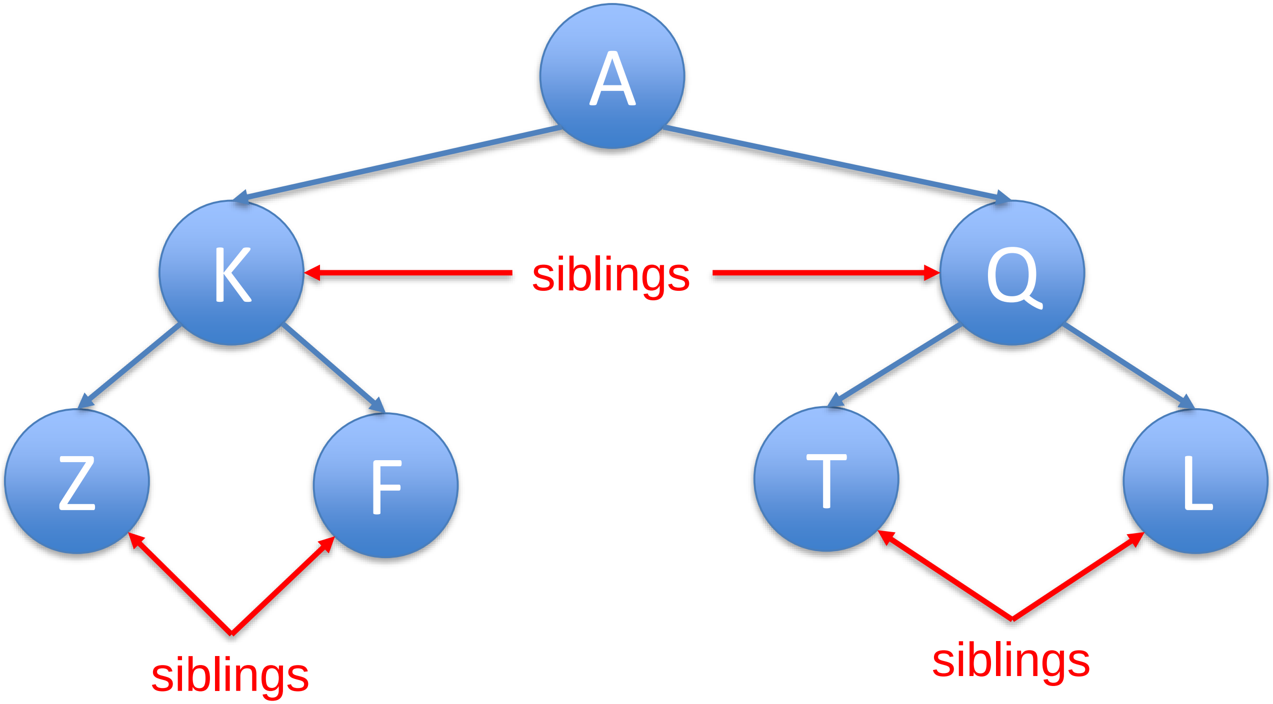
What is a Tree?

- Nodes with no children are called ***leaves***
- Which are leaves?



What is a Tree?

- Nodes with same parent are ***siblings***
- *Which are siblings?*

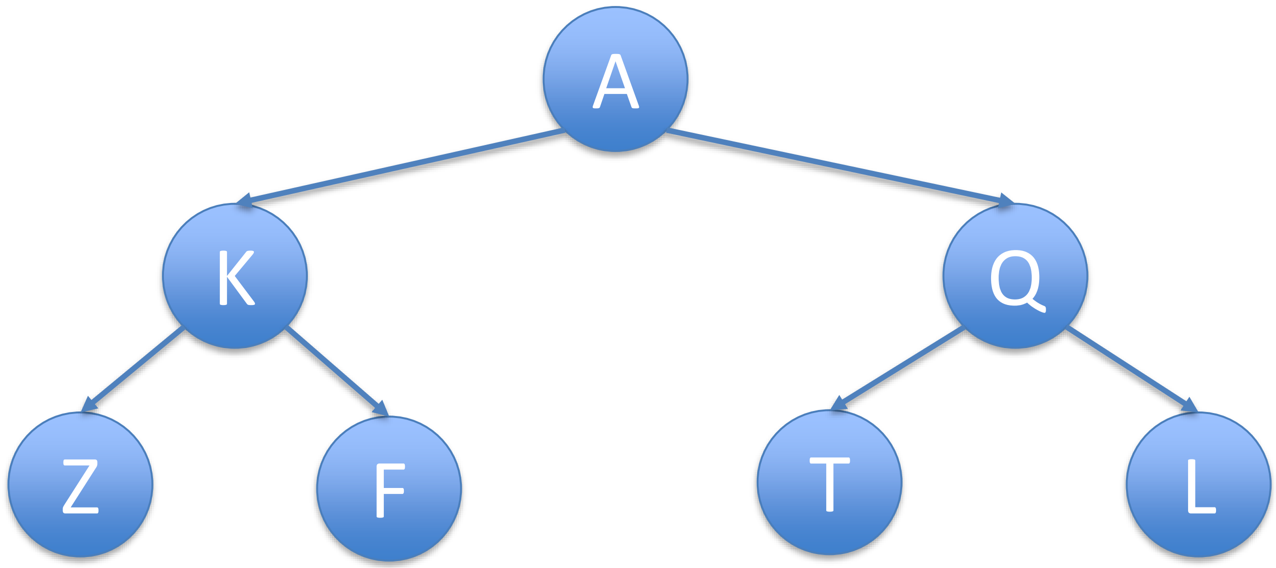


What is a Tree?

- If there is a path between node A and node Z:

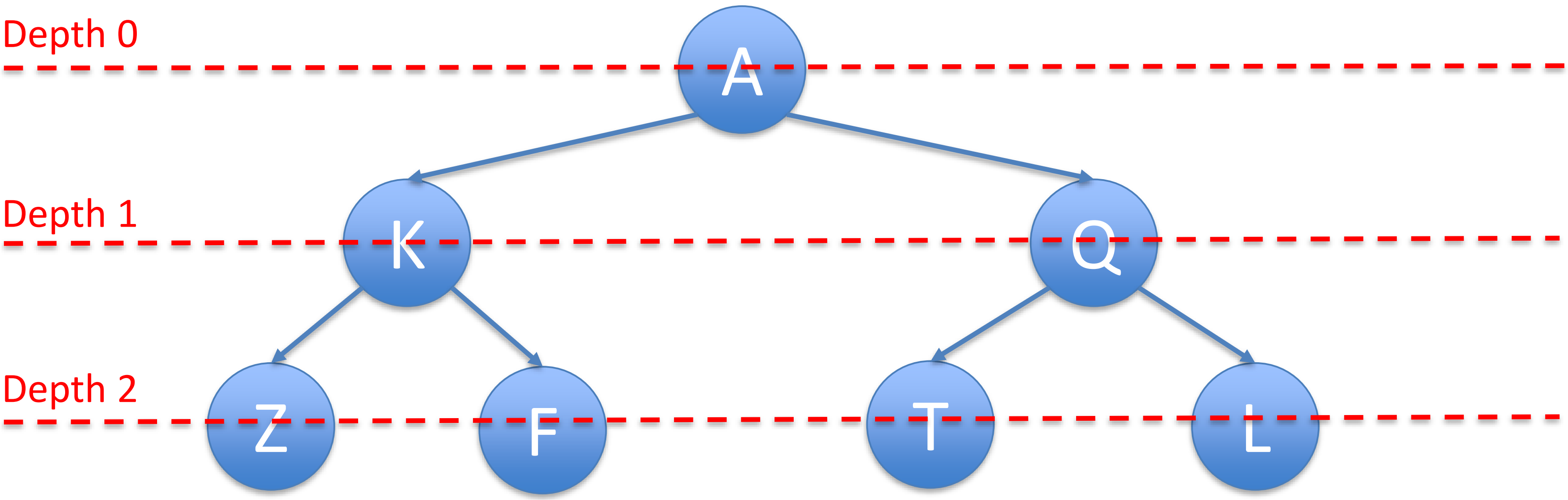
  Z is a ***descendant*** of A

  A is an ***ancestor*** of Z



What is a Tree?

- ***Depth*** of a node: The number of ancestors excluding itself.



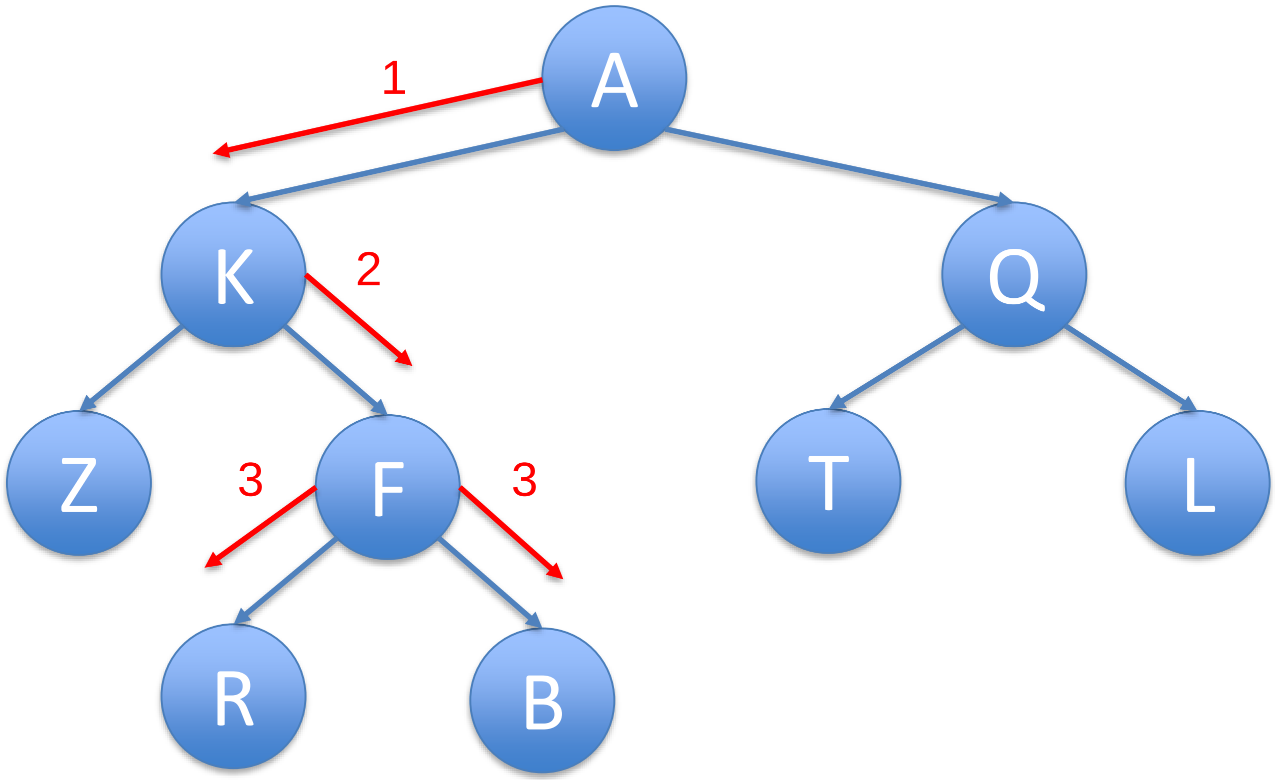
Count number of edges between root and node for depth

What is a Tree?

- *Height* of a tree: Number of edges between root and farthest leaf

What is the height of this tree?

Height = 3

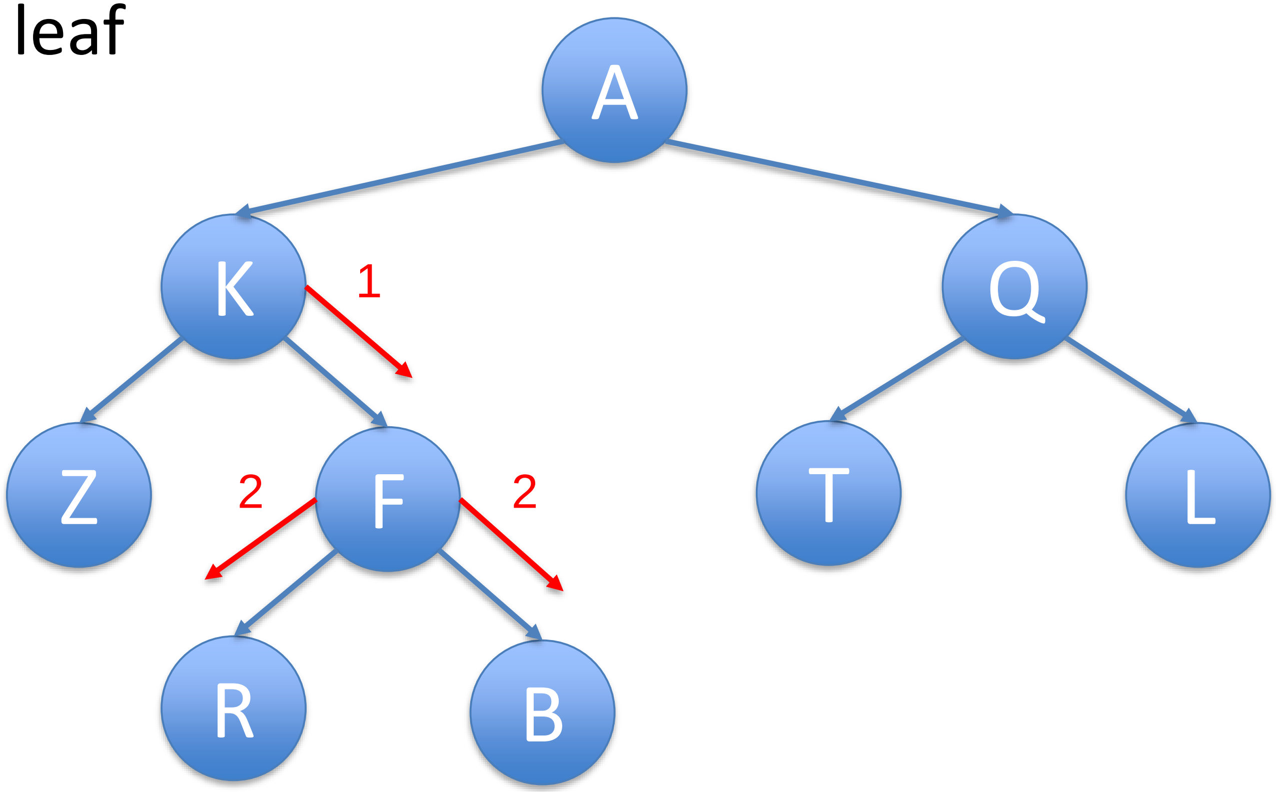


What is a Tree?

- ***Height*** of a node: Number of edges between node and deepest leaf

What is the height of node K?

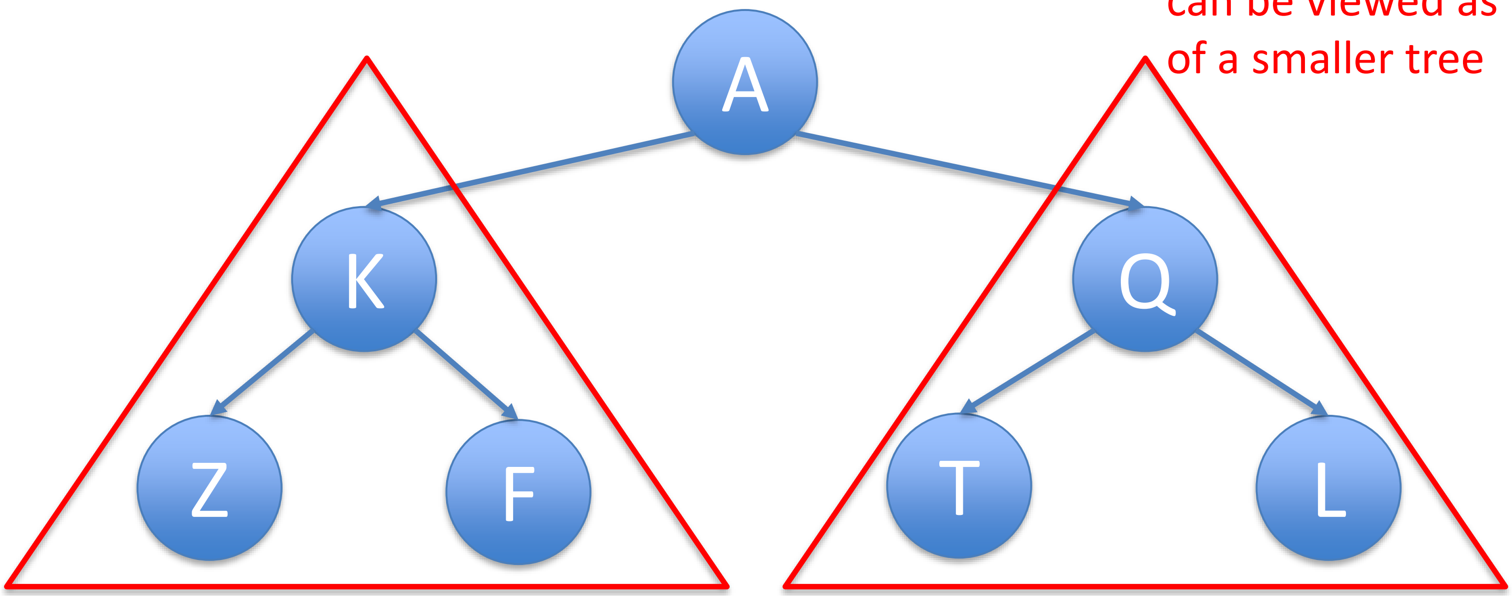
Height = 2



What is a Tree?

- *Subtree*: A tree that consists of a child and the child's descendants

Considered **recursive** because each sub-tree can be viewed as the root of a smaller tree

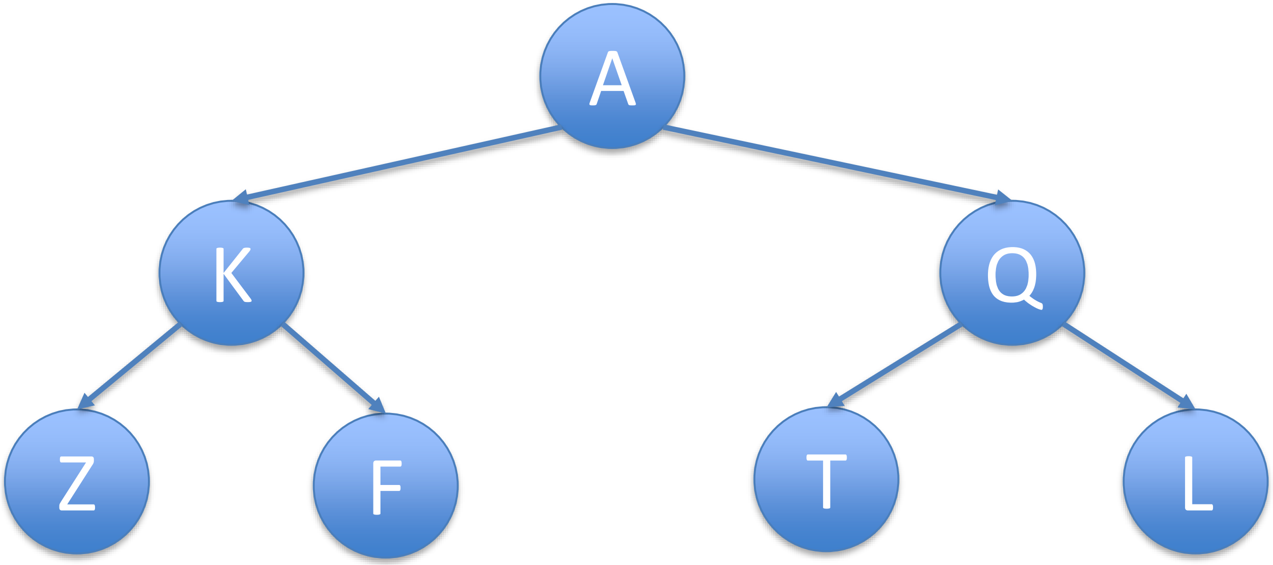


Subtree 1
Includes K, Z, and F

Subtree 2
Includes Q, T, and L

Tree Terminology Practice

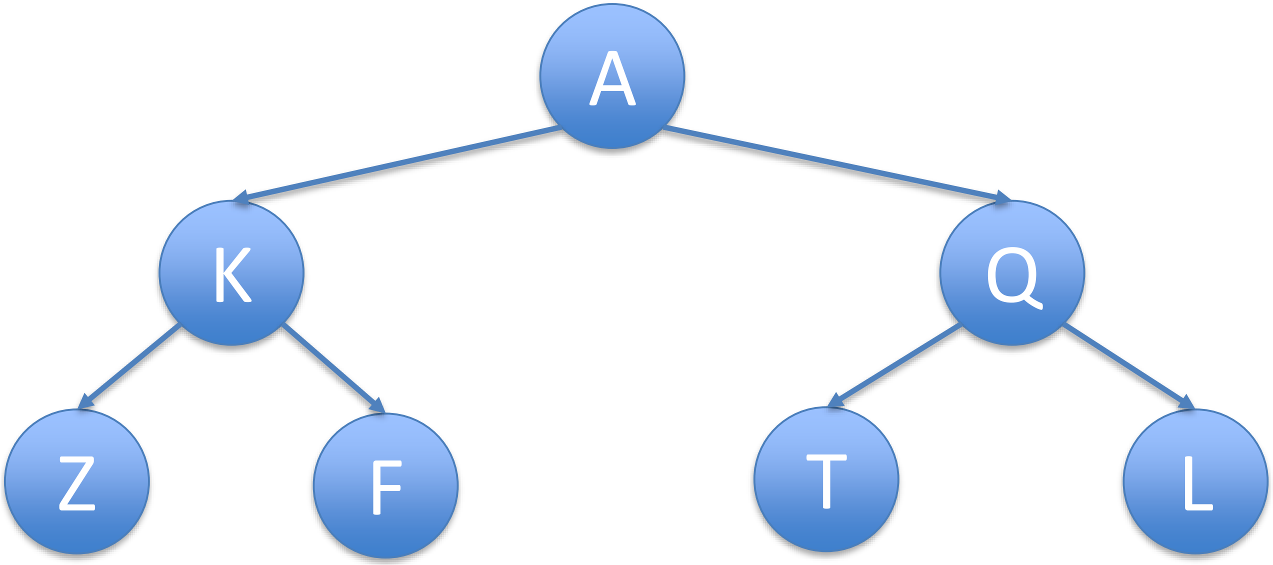
1. How could we describe Z?



Z is a node, a leaf, a sibling of F and a child of K

Tree Terminology Practice

2. How could we describe the relationship between T and L?



T is a sibling of L and they are both leaves

Tree Terminology Summary

- A *tree* is a collection of nodes(elements)
- Each node may have 0 or more *children*
  - (Unlike a list, which has 0 or 1 successors)
- Each node has *exactly one* *parent*
  - Except the starting / top node, called the *root*
- Links from a node to its successors are called *edges* or *branches*
- Nodes with same parent are *siblings*
- Nodes with no children are called *leaves*

Types of Trees

Types of Trees

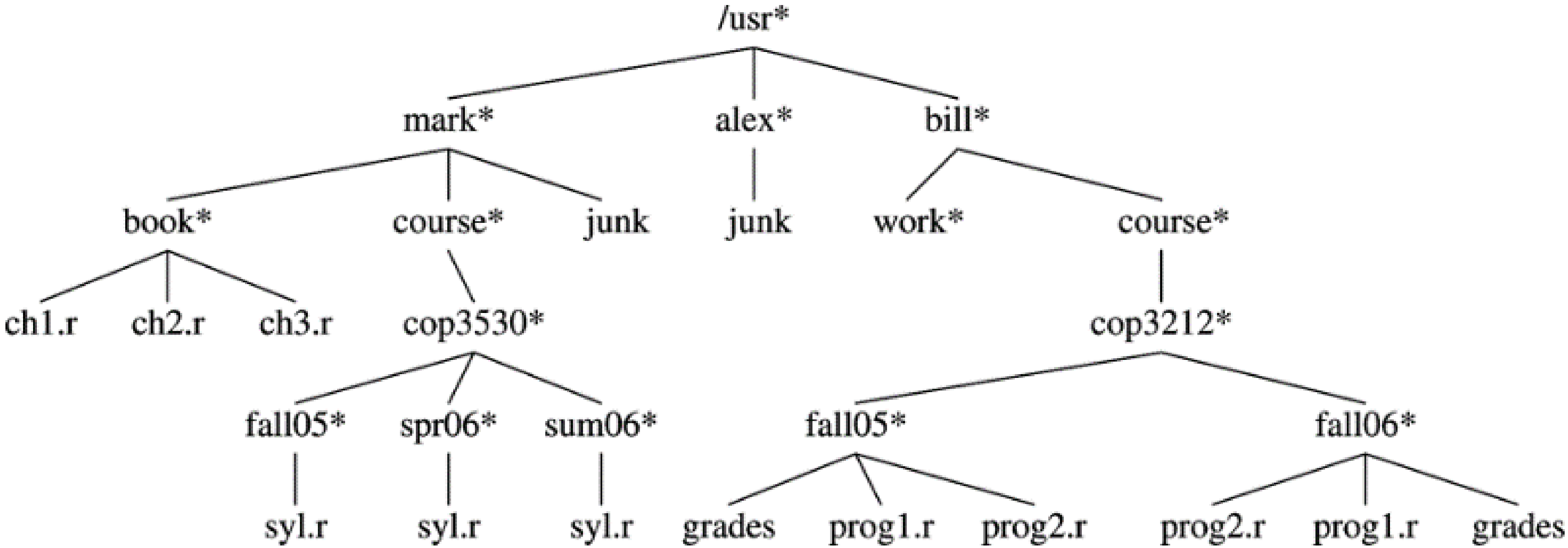
- Regular Tree
- Regular Binary Tree
- Binary Search Tree (BST)

All regular binary trees are also regular trees.

All binary search trees (BST) are also regular binary trees.

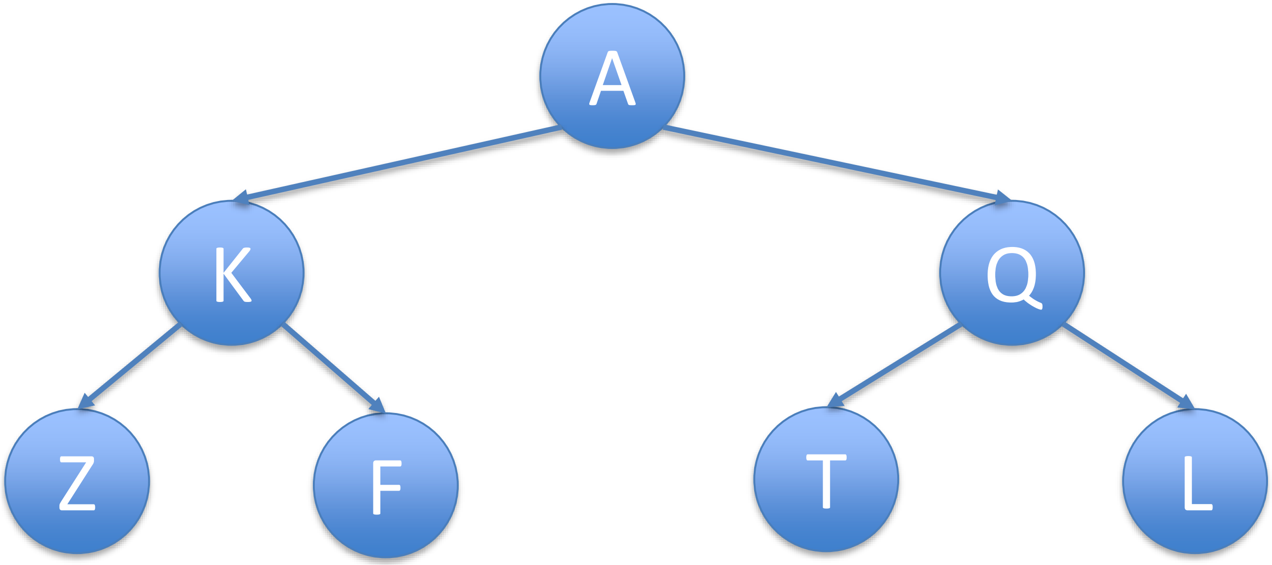
Regular (Non-binary) Tree

- Many links to many children



Regular Binary Tree

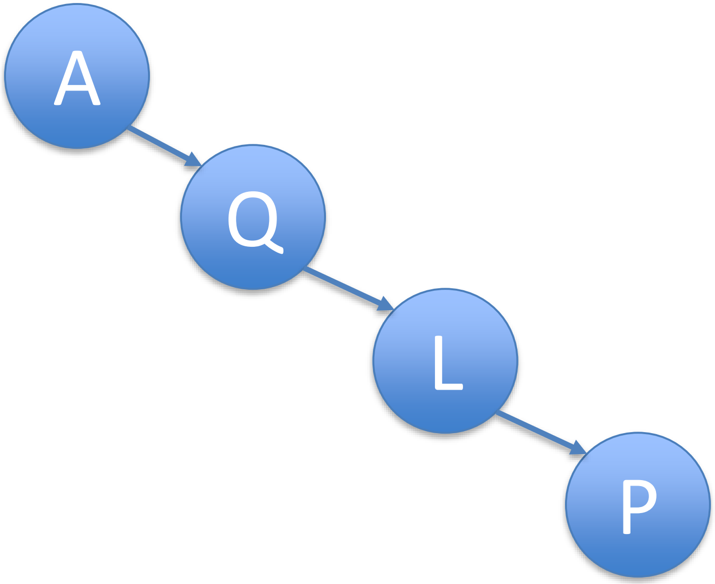
- No node can have more than two children.



Average depth is $O(\sqrt{n})$

Regular Binary Tree

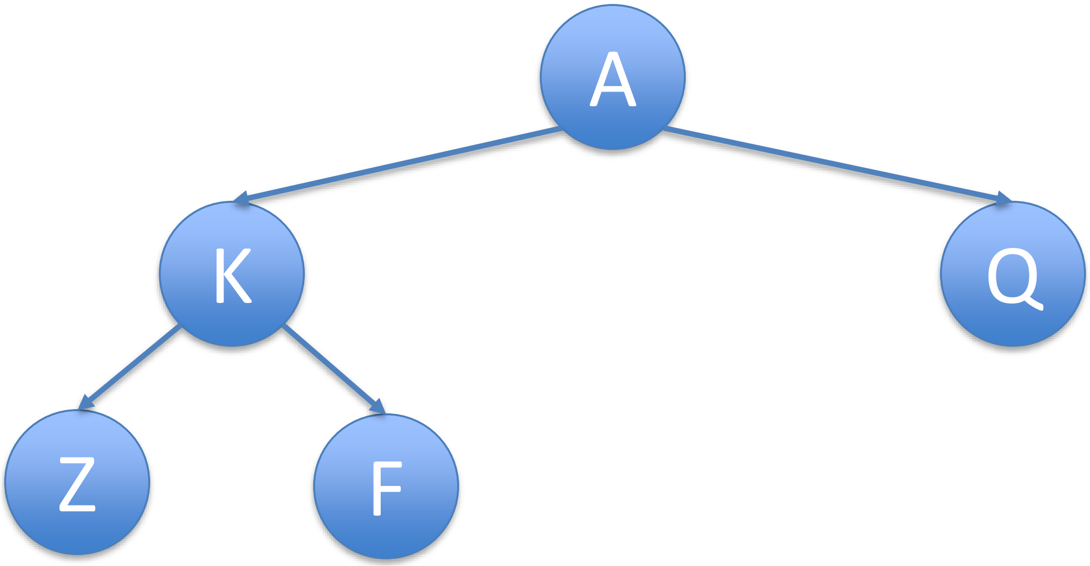
- No node can have more than two children.



Worst scenario depth is $O(n-1)$

Full Binary Tree

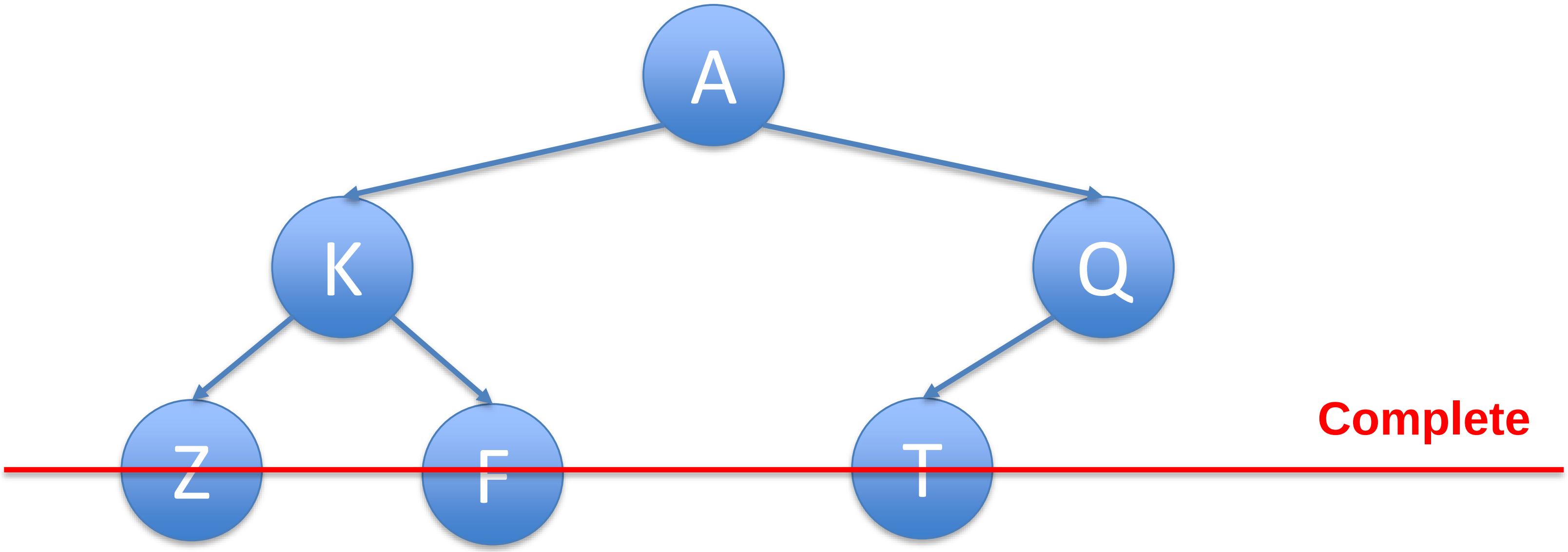
- A binary tree is *full* if all nodes have exactly 0 or 2 child nodes



Full

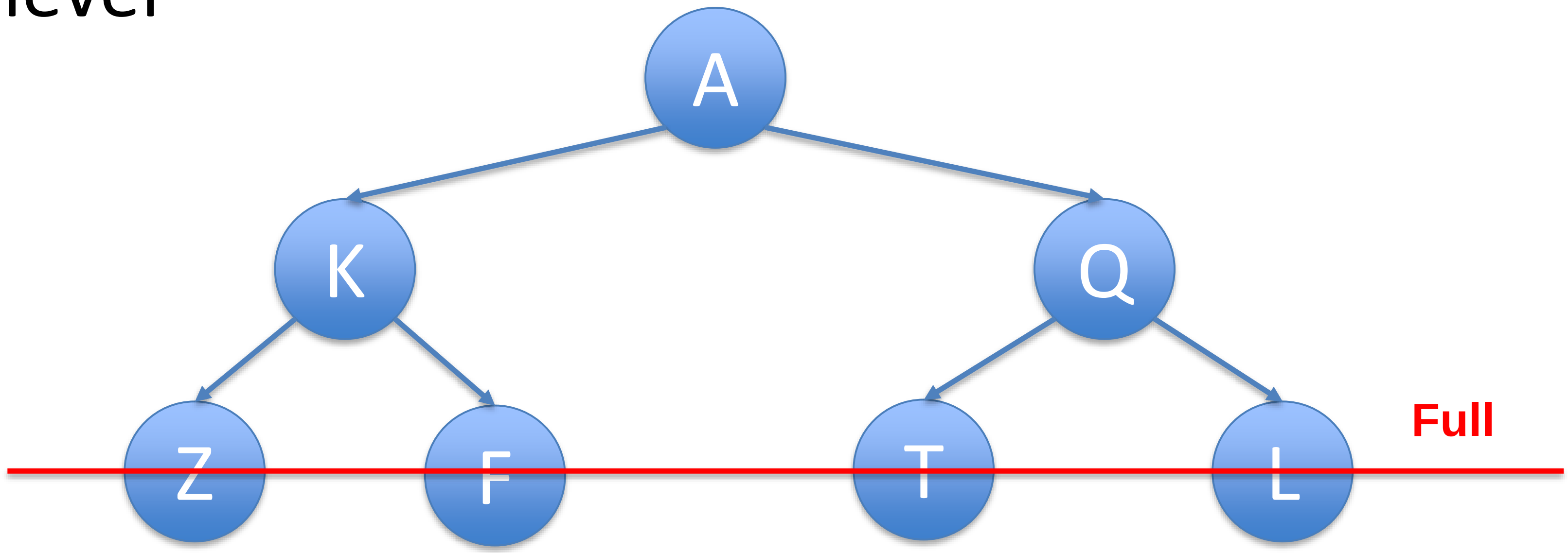
Complete Binary Tree

- A binary tree is *complete* if:
  - Every level but the last must be full
  - All leaves are as far to the left as possible



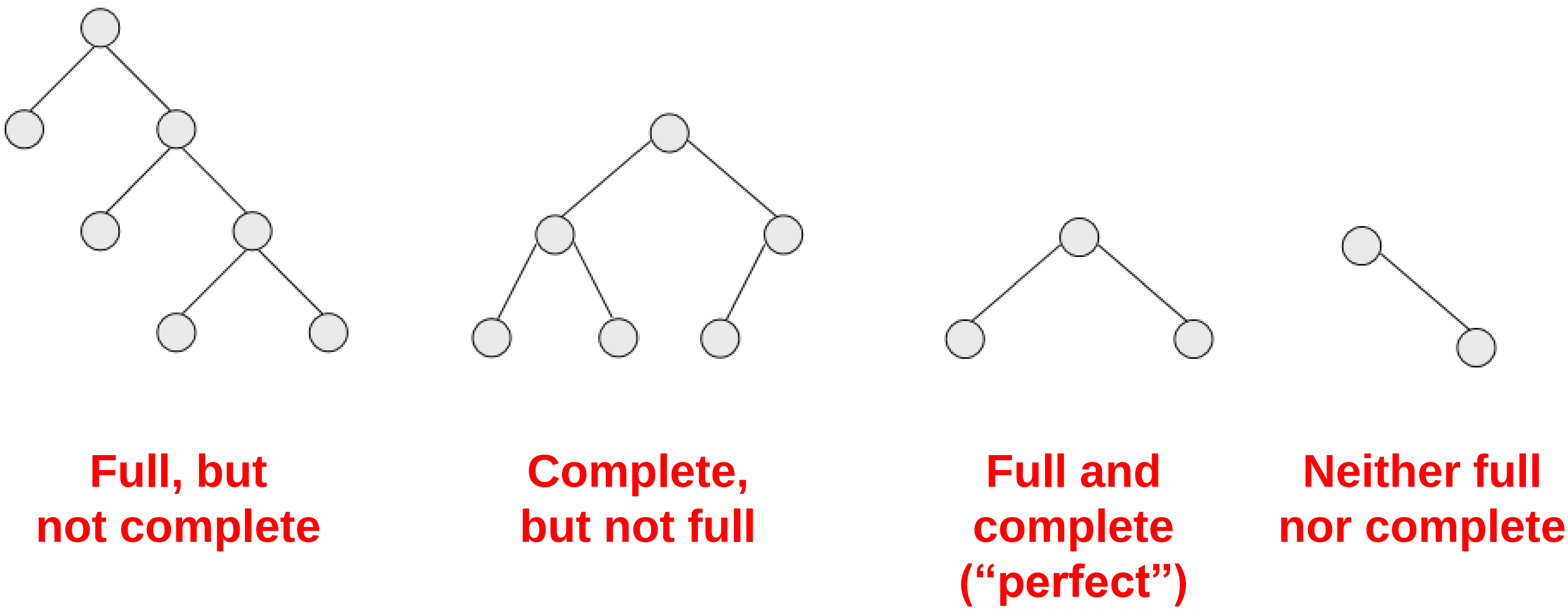
Perfect Binary Tree

- A binary tree is *perfect* if all interior nodes have 2 children and all leaves are at the same level



Complete & Full Binary Trees

- Is each tree full, complete, neither, or both?



Binary Search Tree (BST)

- A **binary search tree (BST)** or **ordered binary tree** is a type of binary tree where the nodes are arranged in order:
  - For each node, all elements in its left subtree are less than the node (<)
  - All the elements in its right subtree are greater than the node (>)

BSTs Next Class!

Other Binary Tree Information

- Trees are ***SHALLOW*** – they can hold many nodes with very few levels
- A height of 19 can hold 1,048,575 nodes

- $2^{(height+1)}$ -1 = How many TOTAL nodes can be held by this tree
  - Can also be expressed as $2^{(depth+1)}$ - 1

Tree Implementations

Tree Implementation

- There are two ways to construct trees
  - Linked Lists
    - Use links to connect to the other nodes in the tree
  - Array (K-ary)
    - Can only use if we know the MAXIMUM number of children allowed

K-ary Trees (also called M-ary)

- “k” is the number of children (links)
- Built as an array of nodes
- Will only work if we know the MAXIMUM number of children
- Empty spots in the array to denote a missing node
- Useful in coding since we can dictate the number of nodes we want
  - Also since there is a formula to calculate the node’s kids
- Child and grandchild index and corresponding items can be found in constant time.

From: http://cs.lmu.edu/~ray/notes/orderedtrees/

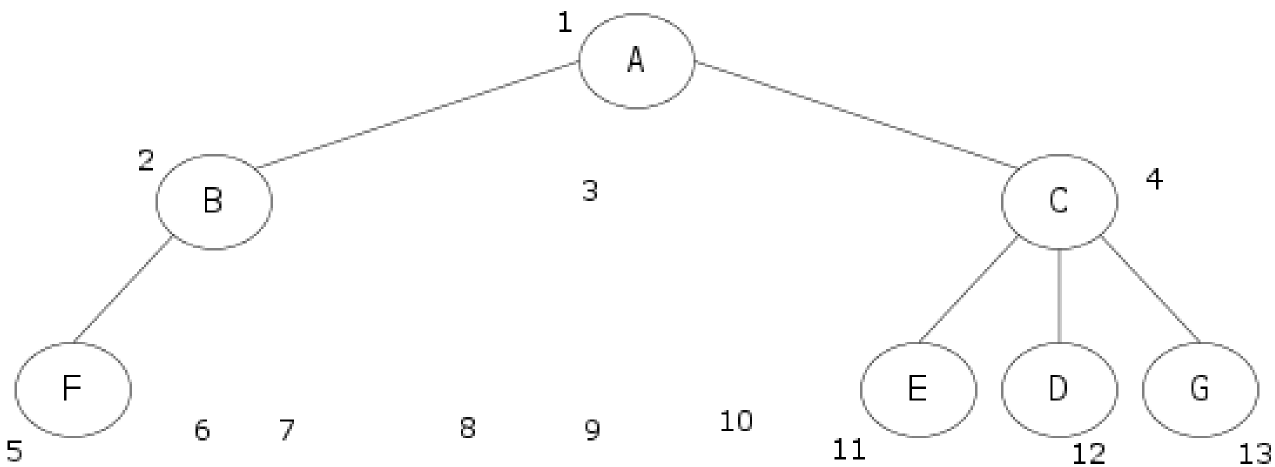
K-ary Trees

- A k-ary tree is a tree in which the children of a node appear at distinct index positions in 0..k-1
- This means the maximum number of children for a node is k

K-ary Trees

- Some k-ary trees have special names
  - 2-ary trees are called **binary trees**
  - 3-ary trees are called **trinary trees** or **ternary trees**
  - 1-ary trees are called **lists**

Array Representation Of A Tree



1	A
2	B
3	
4	C
5	F
6	
7	
8	
9	
10	
11	E
12	D
13	G

From: http://cs.lmu.edu/~ray/notes/orderedtrees/

Array Representation Of A Tree

- For k-ary trees, with first node=1:
  - parent(i) = (i – 2) / k + 1 (0-origin: (i – 1) / k)
    for binary: i / 2
  - child(i) = k (i – 1) + 1 + j (0-origin: k * i + j + 1)
- For binary trees, especially simple:
  - parent(i) = i/2, child(i) = 2i, 2i+1

Tree Traversals

Traversals of Binary Trees

- To iterate over and process the nodes of a tree
  - We *walk* the tree and *visit* the nodes in order
  - This process is called ***tree traversal***
- Three kinds of binary tree traversal:
  - ***Pre***order
  - ***In***order
  - ***Post***order

Traversals of Binary Trees

- *Preorder*: Visit root, traverse left, traverse right
- *Inorder*: Traverse left, visit root, traverse right
- *Postorder*: Traverse left, traverse right, visit root

Algorithm for Preorder Traversal

1. `if` the tree is empty
2. Return
   `else`
3. Visit the root.
4. Preorder traverse the left subtree.
5. Preorder traverse the right subtree.

Algorithm for Inorder Traversal

1. `if` the tree is empty
2. Return
   `else`
3. Inorder traverse the left subtree.
4. Visit the root.
5. Inorder traverse the right subtree.

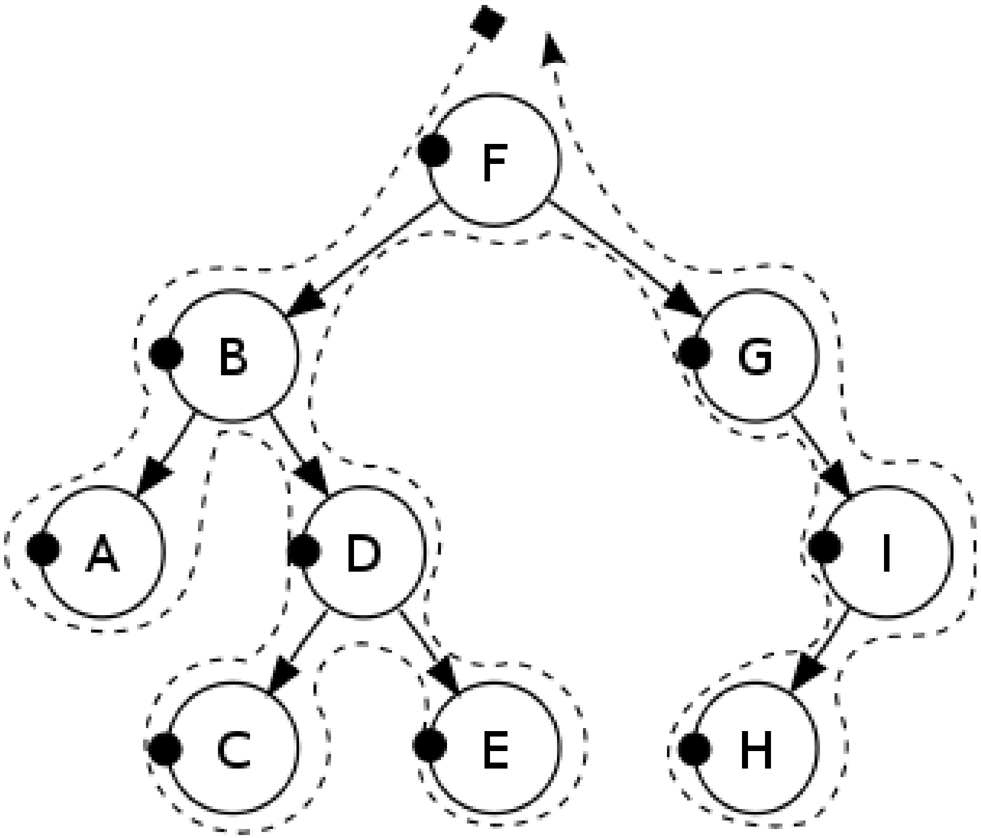
Algorithm for Postorder Traversal

1. `if` the tree is empty
2. Return
   `else`
3. Postorder traverse the left subtree.
4. Postorder traverse the right subtree.
5. Visit the root.

Preorder Traversals

Preorder:
F, B, A, D, C, E, G, I, H

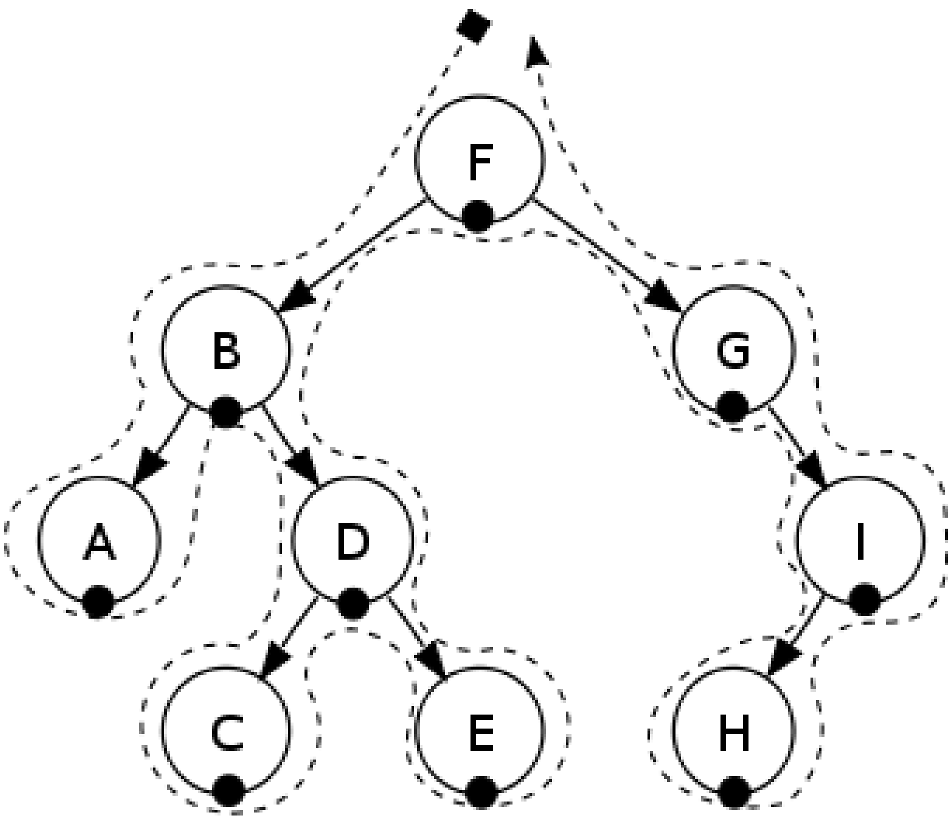
Display a node's data as soon as you see it



1. Display the data part of root element (or current element)
2. Traverse the left subtree by recursively calling the pre-order function.
3. Traverse the right subtree by recursively calling the pre-order function.

Inorder Traversals

Inorder:
A, B, C, D, E, F, G, H, I



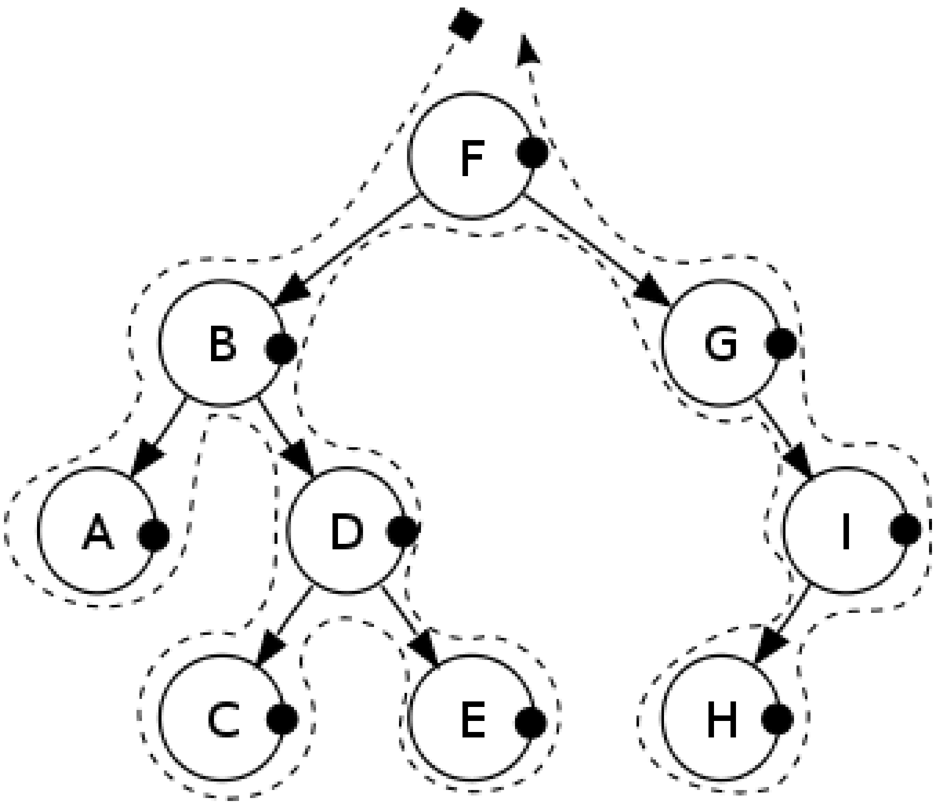
Display the nodes in order (sort of from left to right, with the lower nodes first)

1. Traverse the left subtree by recursively calling the in-order function
2. Display the data part of root element (or current element)
3. Traverse the right subtree by recursively calling the in-order function

From: https://en.wikipedia.org/wiki/Tree_traversal

Postorder Traversals

Postorder:
A, C, E, D, B, H, I, G, F

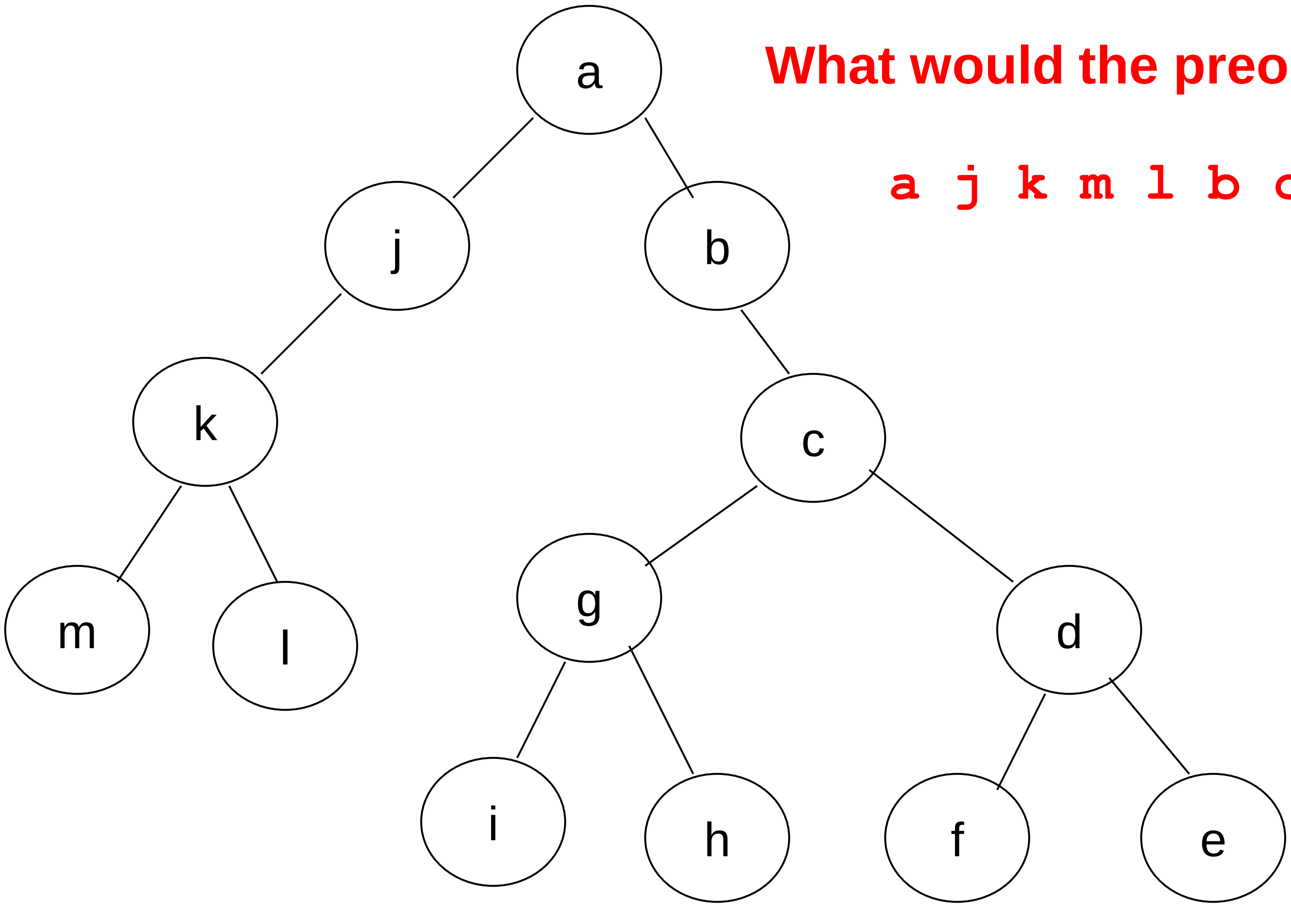


Display a node's data the last time you see it

1. Traverse the left subtree by recursively calling the post-order function.
2. Traverse the right subtree by recursively calling the post-order function.
3. Display the data part of root element (or current element).

From: https://en.wikipedia.org/wiki/Tree_traversal

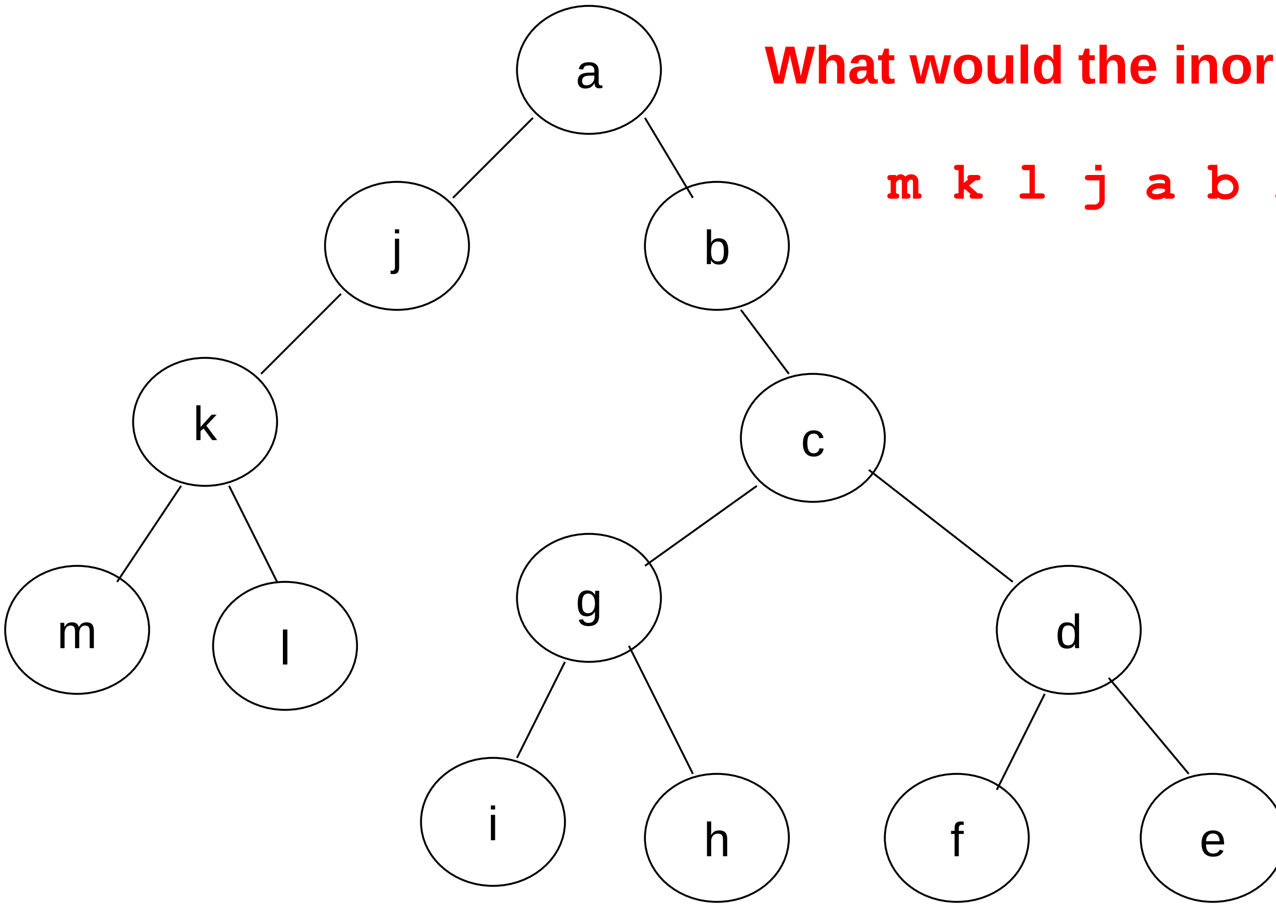
Tree Traversal Example



What would the preorder traversal look like?

`a j k m l b c g i h d f e`

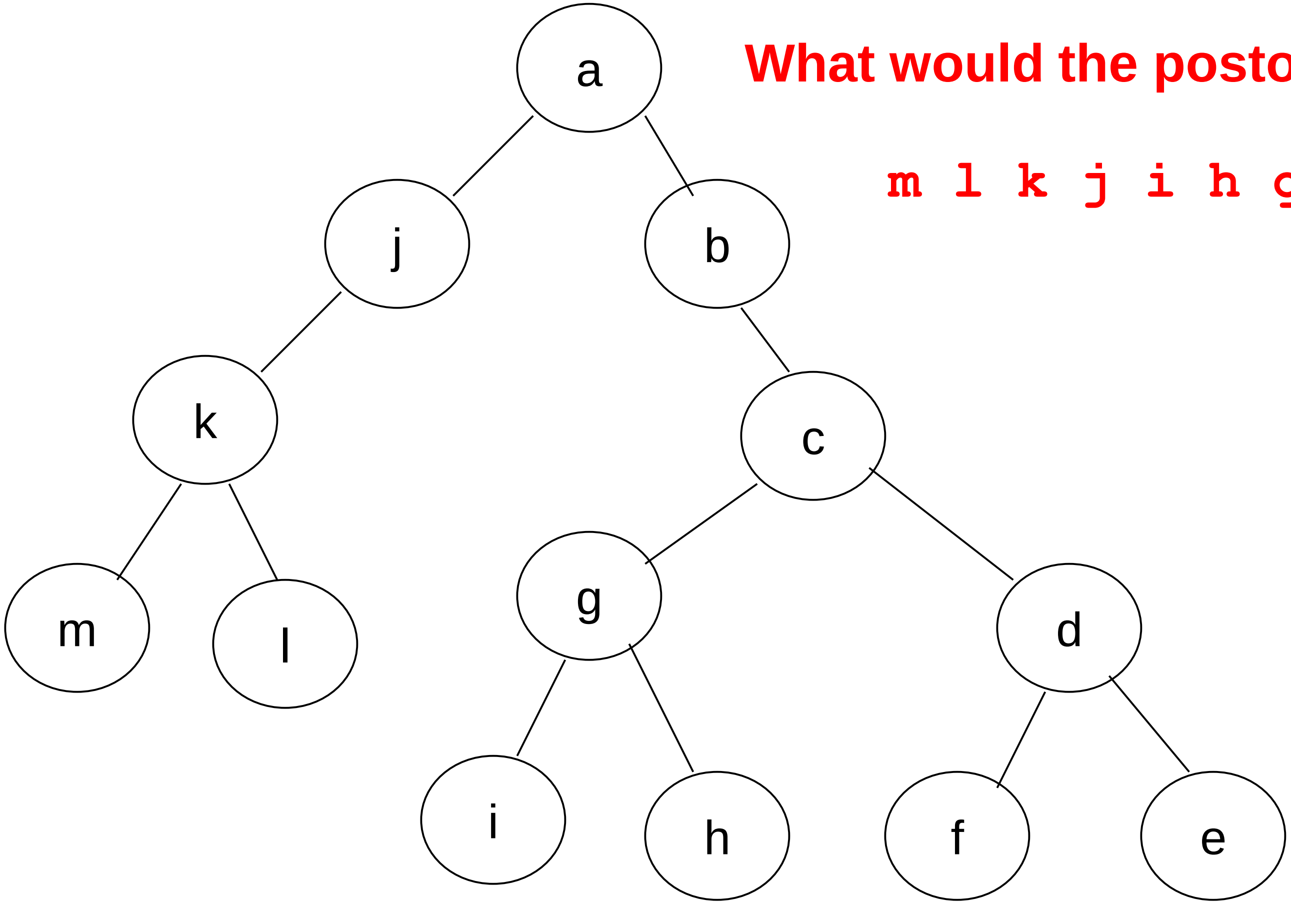
Tree Traversal Example



What would the inorder traversal look like?

`m k l j a b i g h c f d e`

Tree Traversal Example



What would the postorder traversal look like?

m l k j i h g f e d c b a

Preorder Traversals

```
preorder (Node t)
    if (t == null)
        return;

    visit (t.value());
    preorder (t.lchild());
    preorder (t.rchild());

} // preorder
```

Preorder

N L R

Inorder Traversals

```
inorder (Node t)
   if (t == null)
      return;

   inorder (t.lchild());
   visit (t.value());
   inorder (t.rchild());

} // inorder
```

Inorder

L N R

Postorder Traversals

```
postorder (Node t)
    if (t == null)
        return;

    postorder (t.lchild());
    postorder (t.rchild());
    visit (t.value());

} // postorder
```

Postorder

L R N

Another Tree Traversal

- A *level-order* walk iterates over all the nodes level-by-level, starting from the root (level 0)– this is known as a *breadth-first search*
- Nodes are traversed level by level
  - Root node is visited first
  - Followed by its direct child nodes
  - Followed by its grandchild nodes
  - Until all nodes in the tree have been traversed

Tree Functions

Binary Tree Functions

Node Setup

`void insert( x )`	--> Insert x
`void remove( x )`	--> Remove x
`boolean contains( x )`	--> Return true if x is present
`Comparable findMin( )`	--> Return smallest item
`Comparable findMax( )`	--> Return largest item
`boolean isEmpty( )`	--> Return true if empty; else false
`void makeEmpty( )`	--> Remove all items
`void printTree( )`	--> Print tree in sorted order

Generic Struct for Binary Tree

```
private struct BinaryNode
{

    Comparable element; // Data in the node
    BinaryNode *left;   // Left child
    BinaryNode *right;  // Right child

    // Constructors
    BinaryNode(const Comparable & theElement,
               BinaryNode *lt, BinaryNode *rt )
    {
        element  = theElement;
        left     = lt;
        right    = rt;
    }
}
```