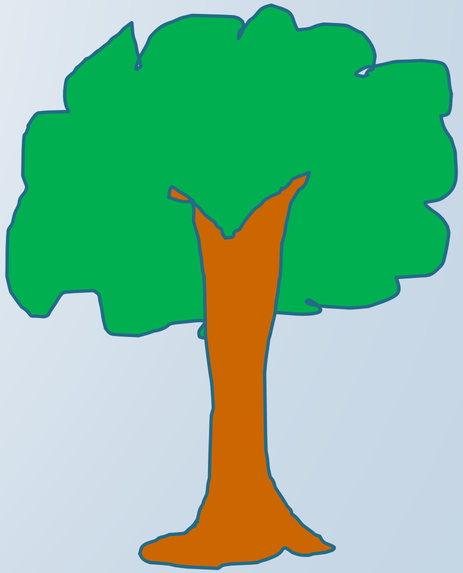
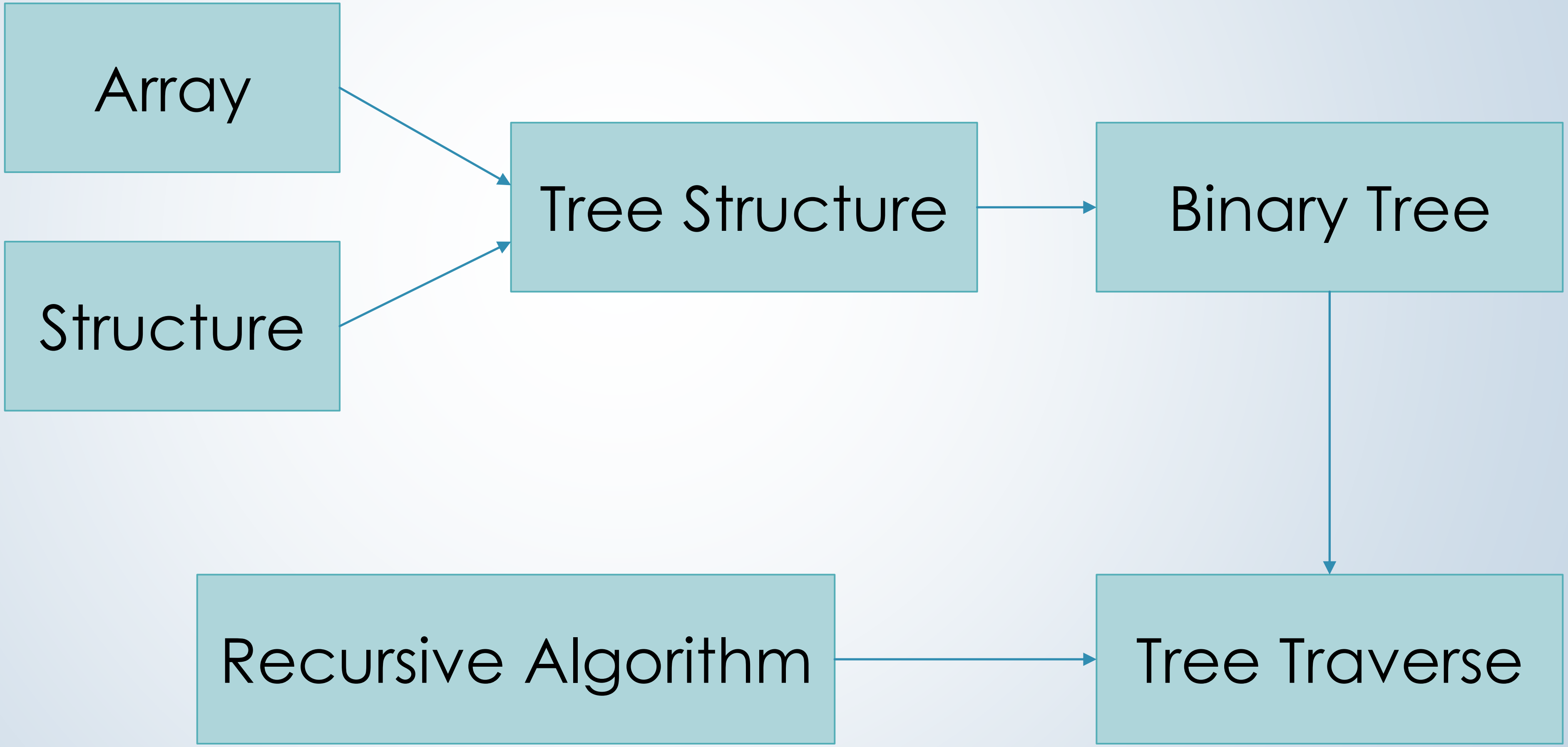
DET102 Data Structures and Algorithms

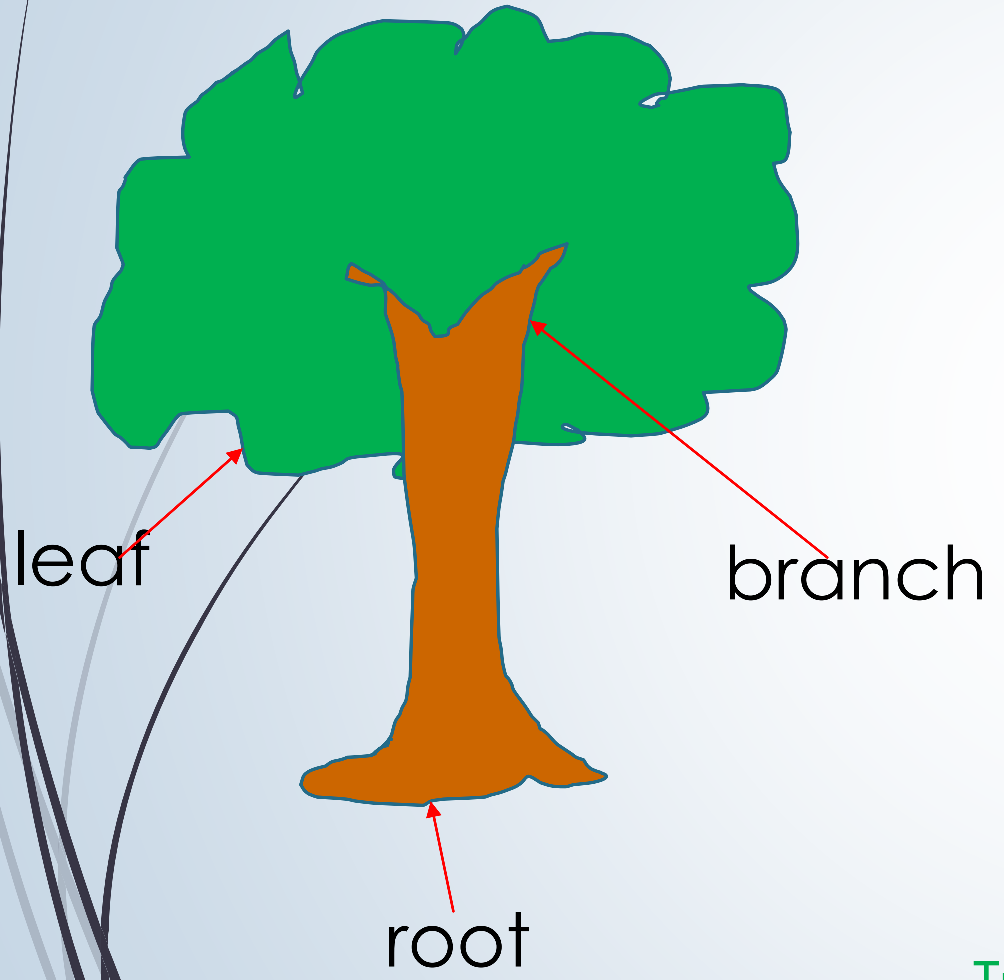
Lecture 06: Tree



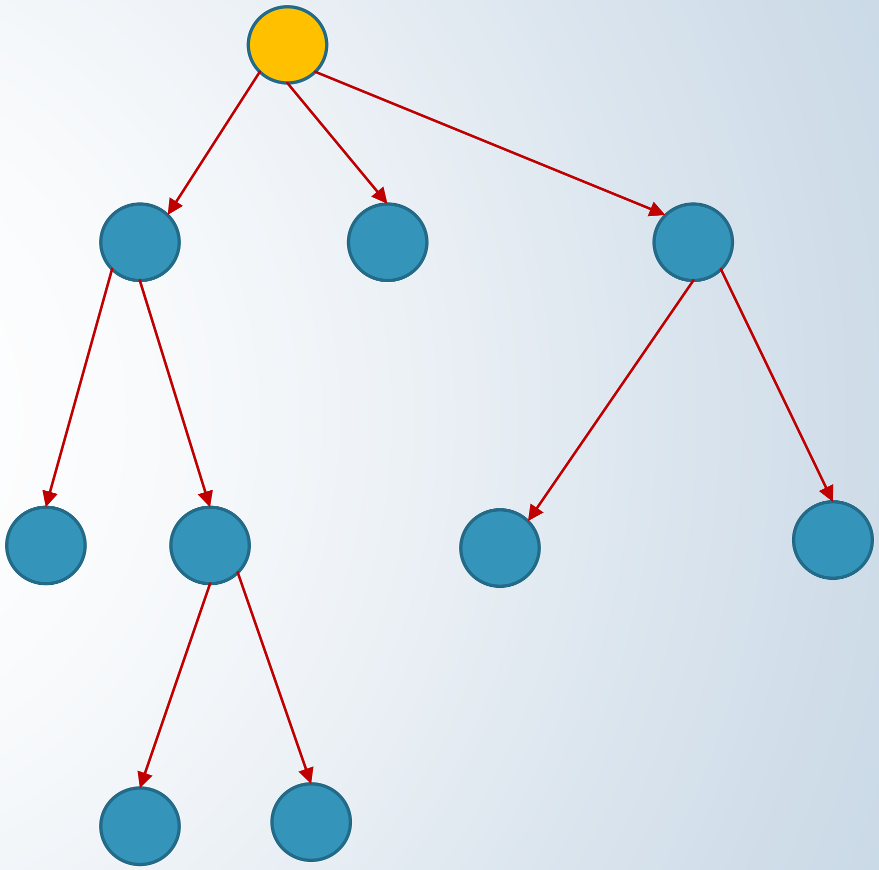
Outline


Array
Structure
Tree Structure
Binary Tree
Tree Traverse
Recursive Algorithm

tree

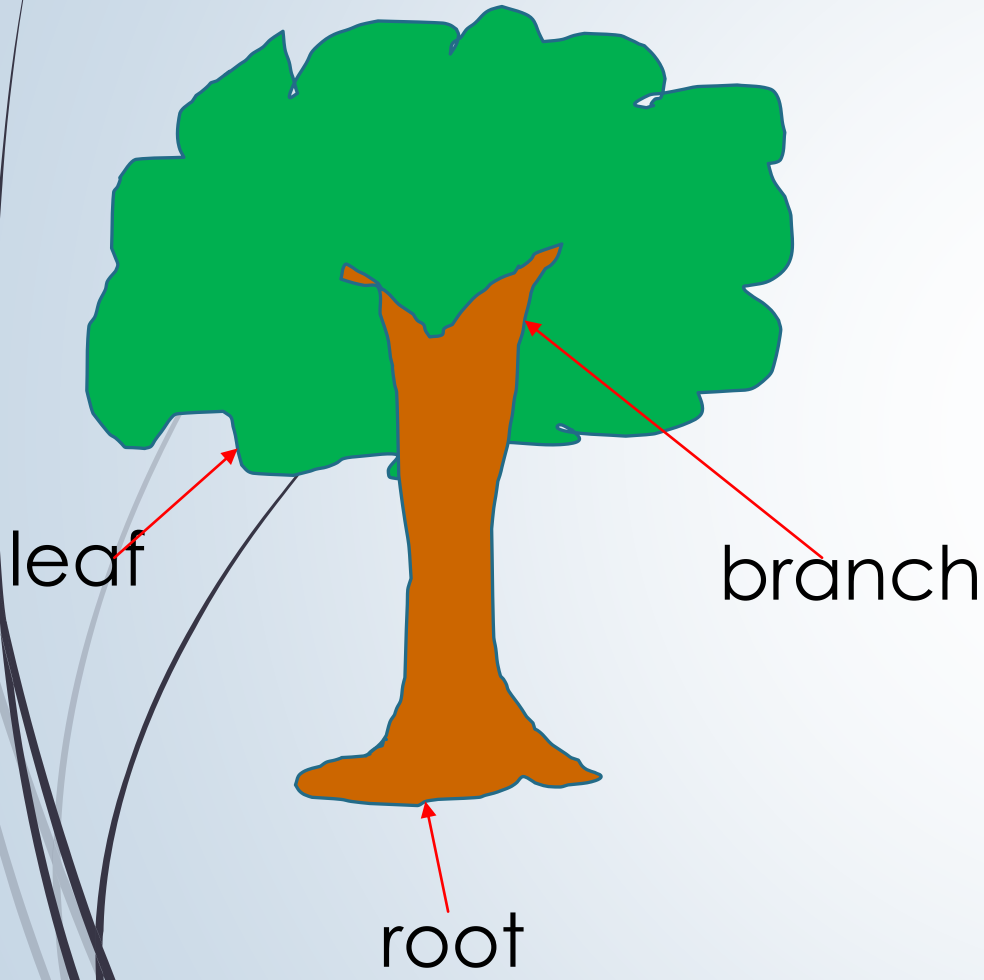


(rooted) tree

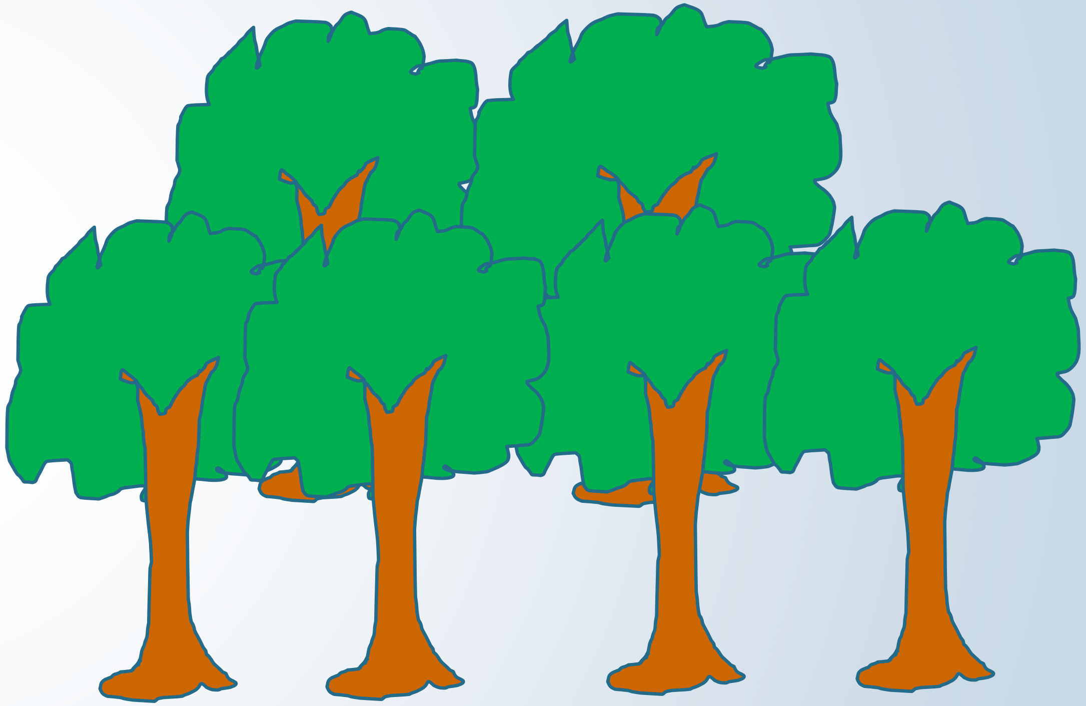


Tree is a data structure, containing nodes and edges.

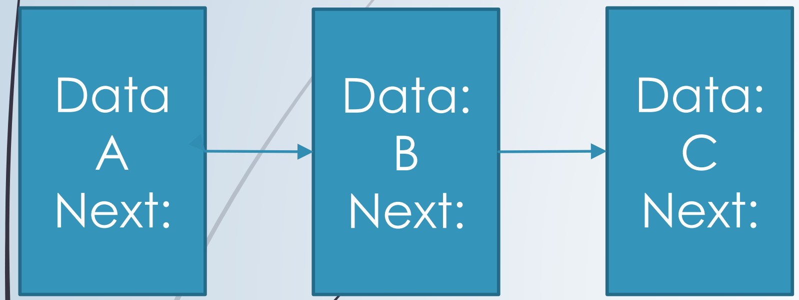
tree



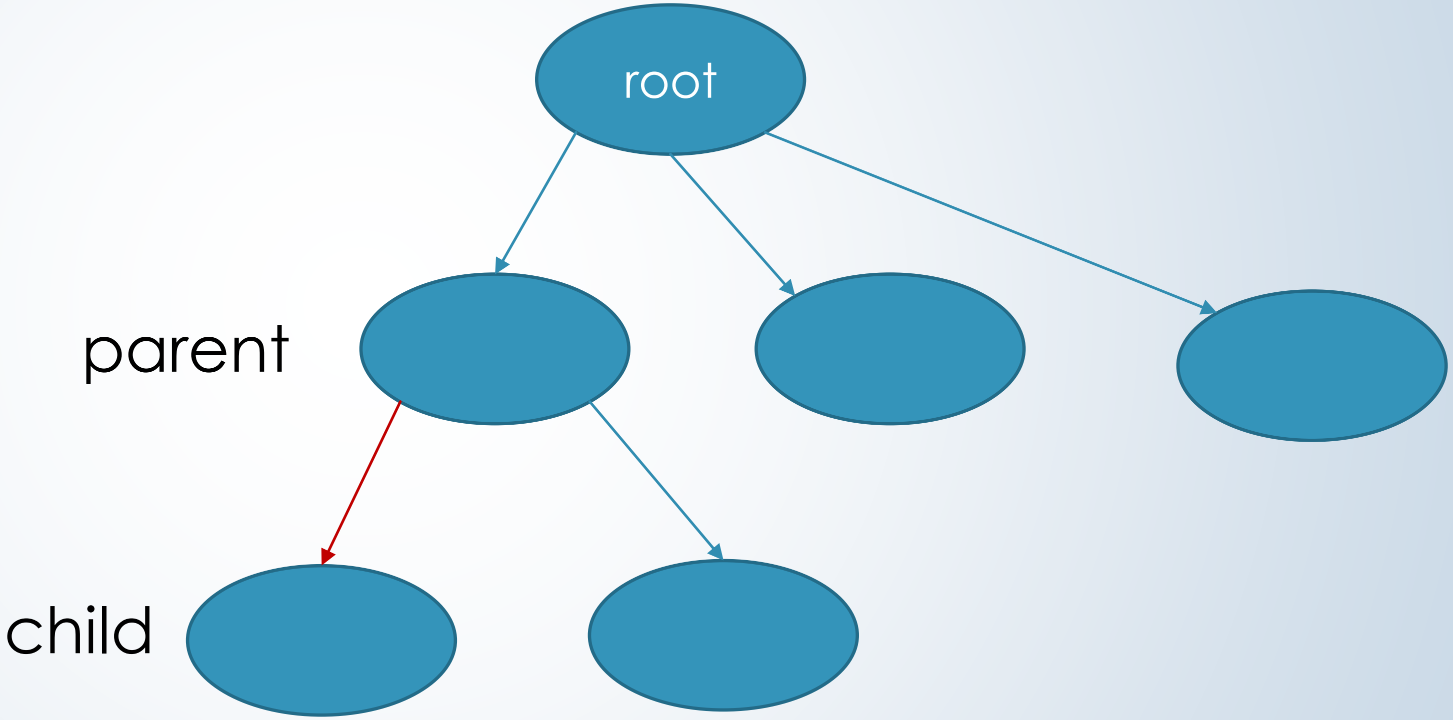
forest



Linked list



Tree



They are siblings.

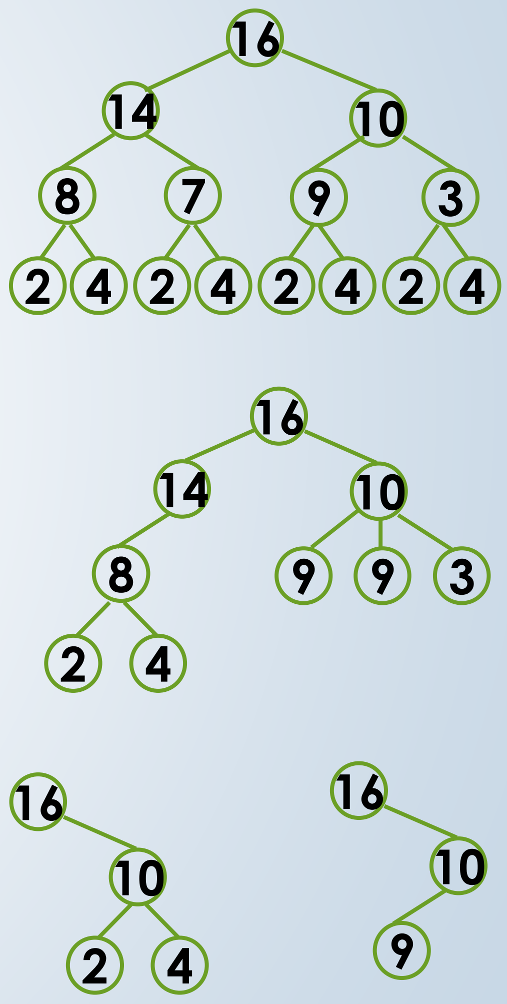
Constraints

one and only one path (unique path)

- 1. Connected
  - Starting from the root, there exists a path reaching each node
- 2. NO cycle
  - Cycle: you will visit some nodes twice

Definition and Terminology

4 examples



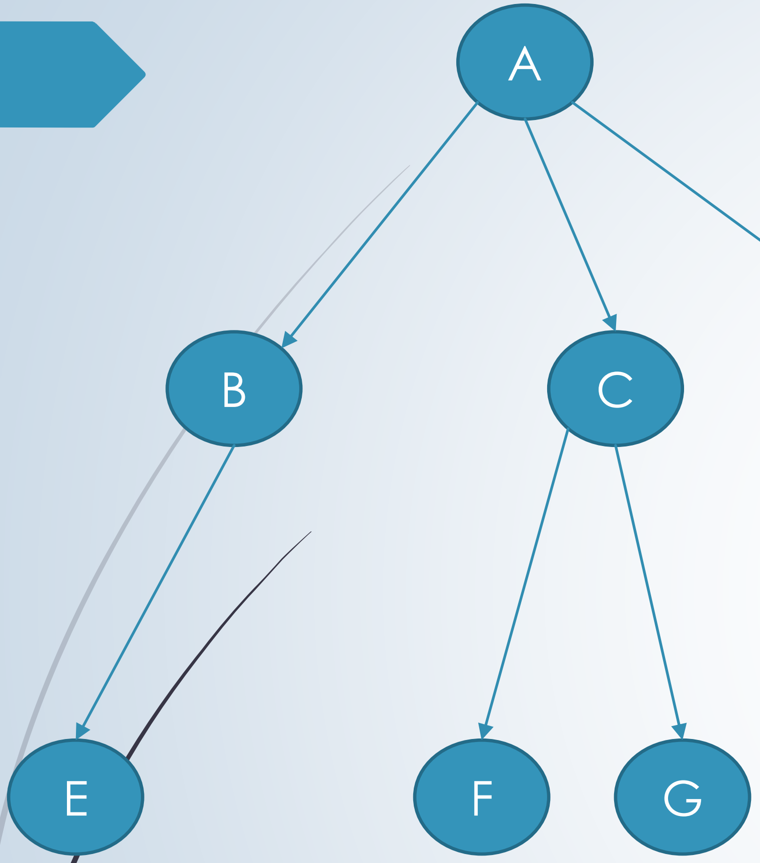
Definition:

Tree is defined as a finite set *T* of one or more nodes such that

a) there is one specially designated node called the **root** of the tree, root(T) and

b) the remaining nodes (excluding the root) are partitioned into $m \geq 0$ disjoint sets $T_1, T_2, \ldots, T_m$ and each of these sets in turn is a tree. The trees $T_1, T_2, \ldots, T_m$ are called the **subtrees** of the root.

A is the root.



Level 1

Level 2

Level 3

Parent-child relationship

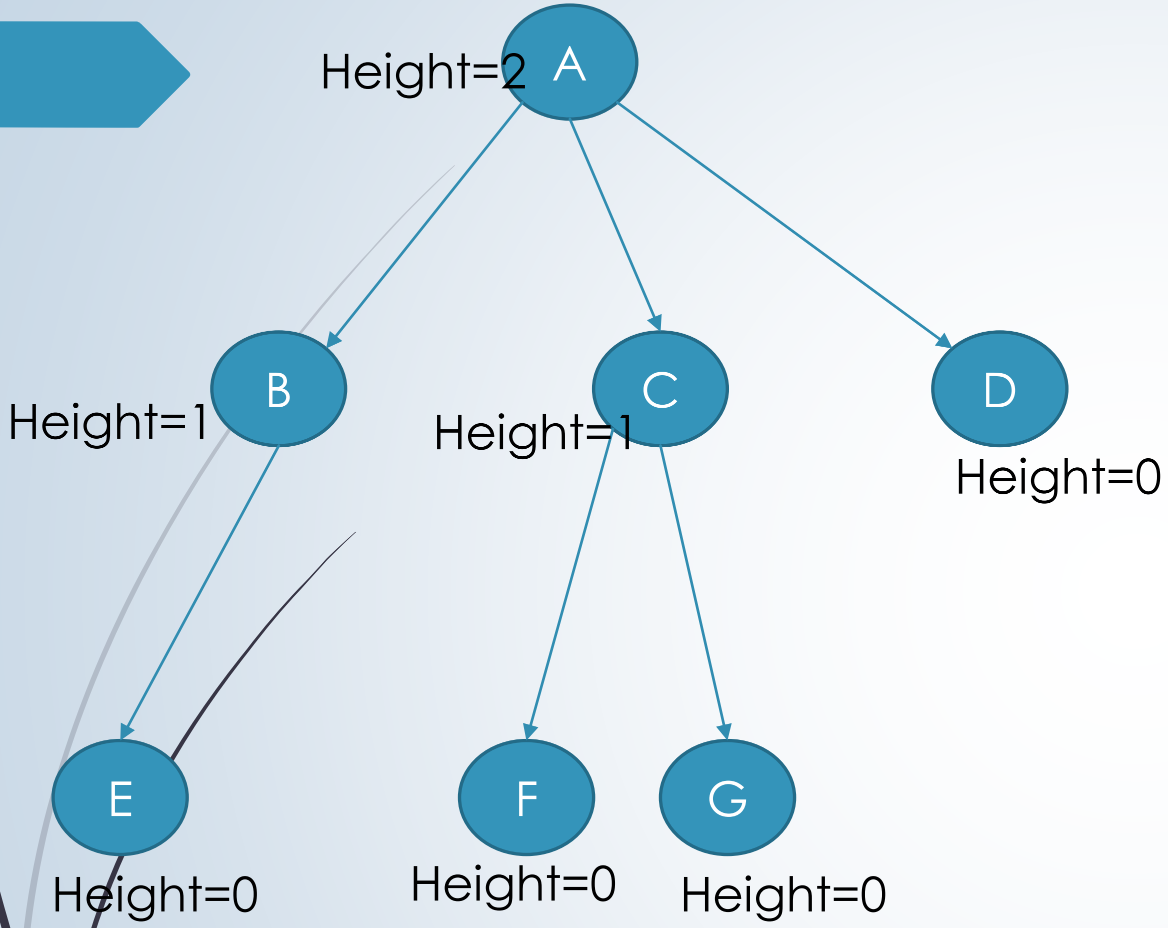
Question 1: Who is(are) node C's sibling(s) ?

Question 2: Who is(are) node F's sibling(s) ?

The ***level*** of a node is defined by 1+ the number of connections between the node and the root.

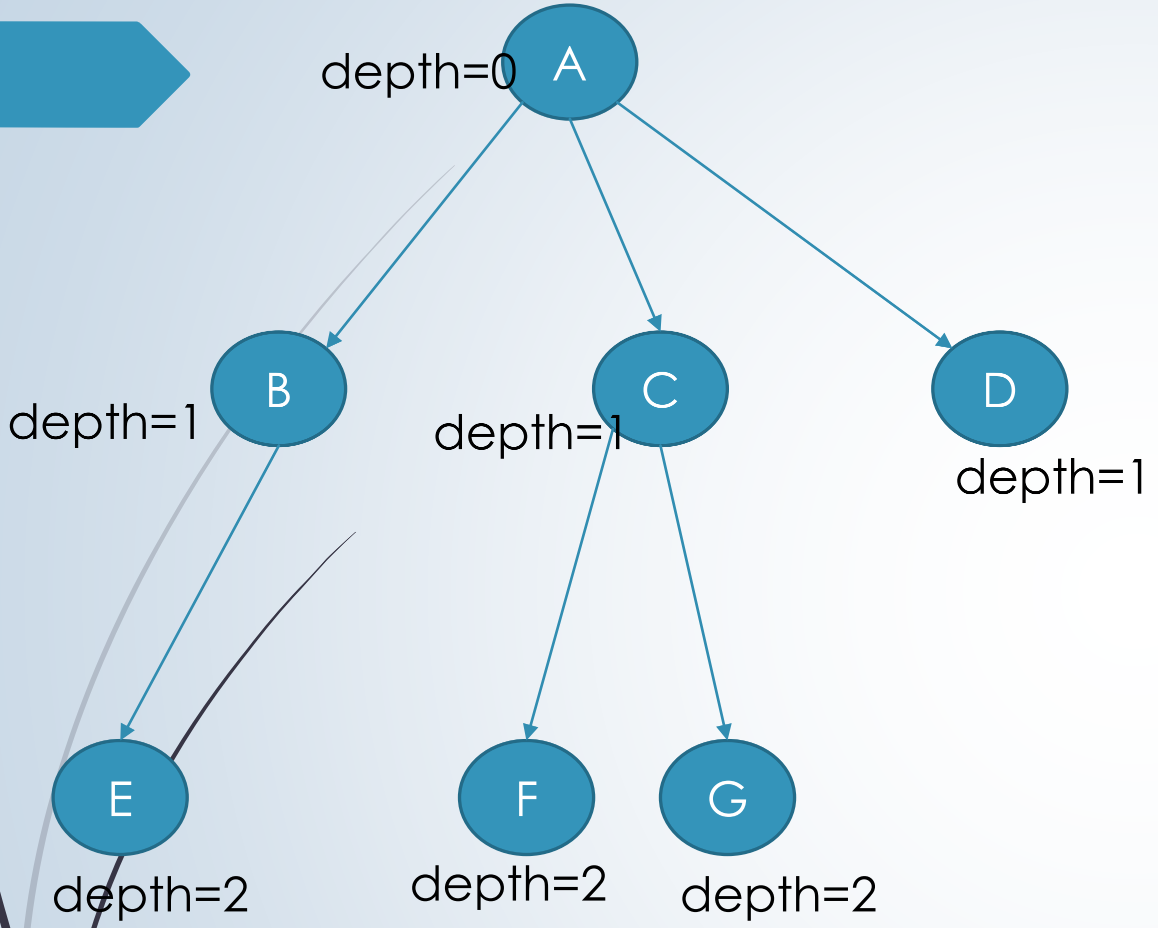
Each child has only one parent. Root has no parent.

Siblings: nodes having the same parent.



Height of a node is the number of edges between it and the furthest leave on the tree.

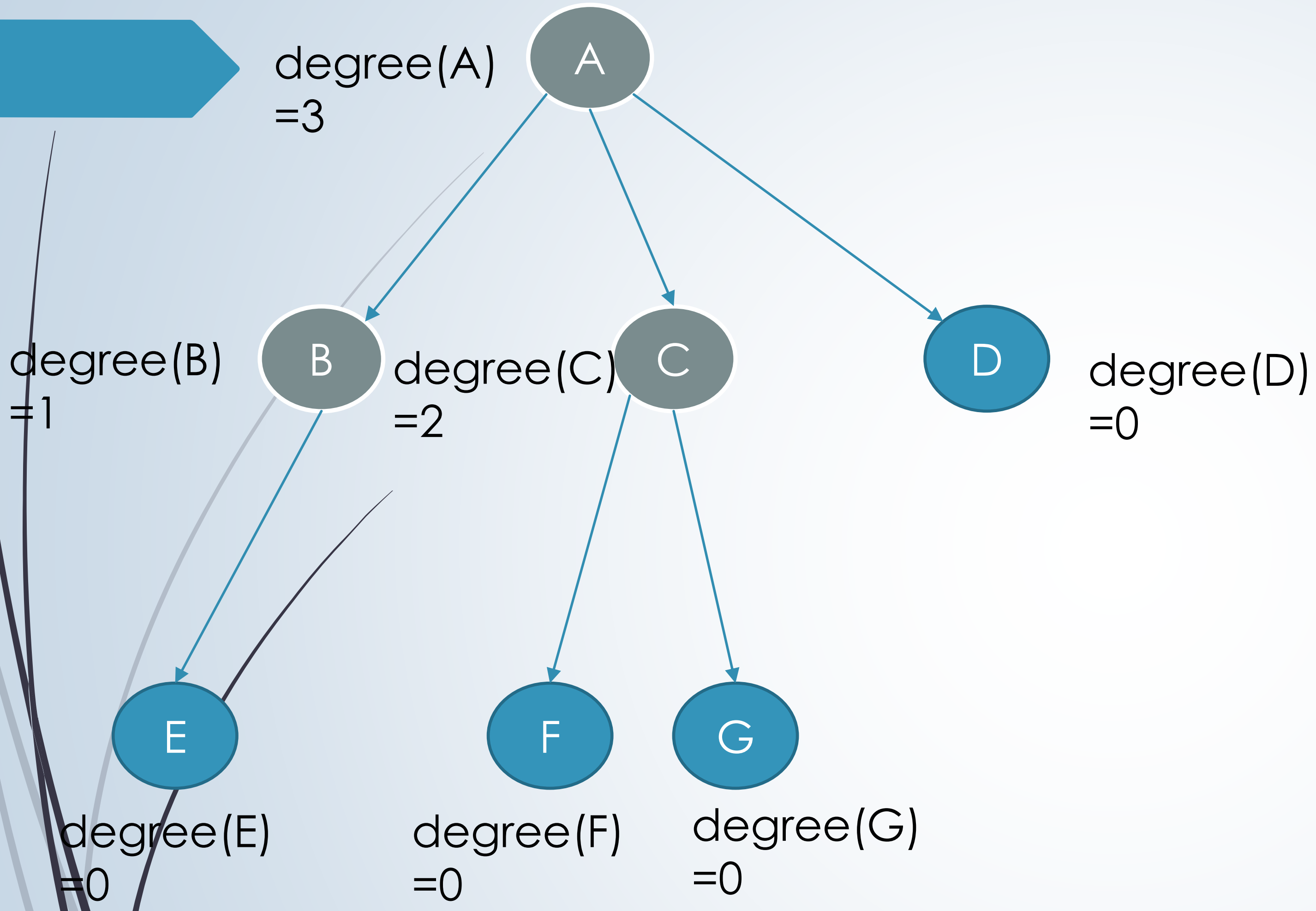
Height of the tree is the height of the root.



level = depth +1

Depth of a node is the number of edges between it and the root.

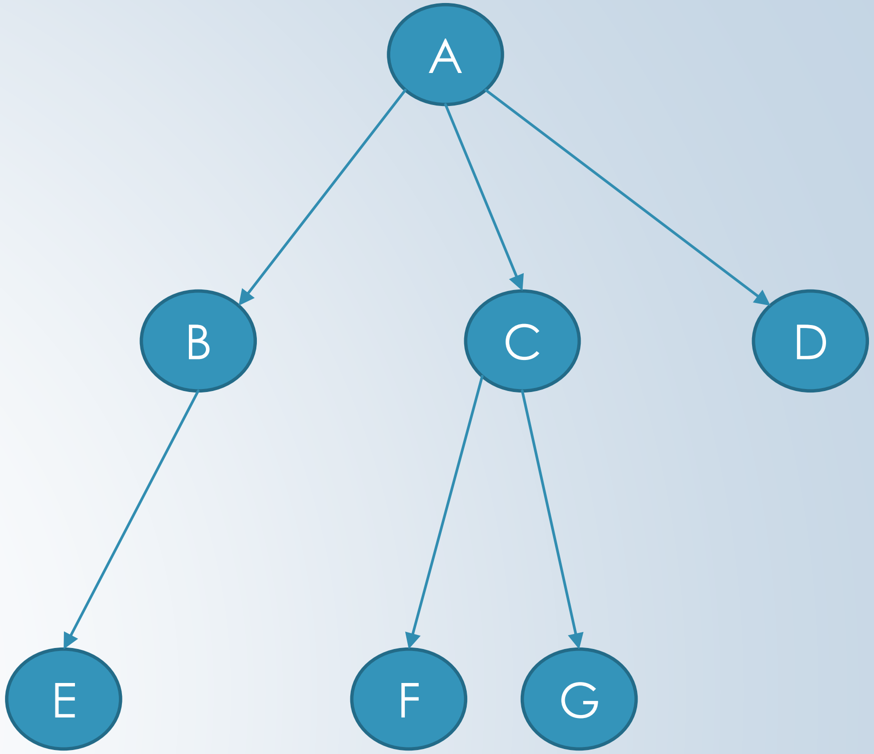
Height and depth move inversely.



Degree of a node is the number of children of that node.

Degree of an internal node is at least 1.
Degree of a leaf is zero.

Terminology	Explanation
Degree of a node	The number of subtrees of a node
Terminal node or **leaf**	A node of degree zero
Branch node or internal node:	A nonterminal node
Parent and **Siblings**	Each root is said to be the parent of the roots of its subtrees, and the latter are said to be siblings; they are children of their parent.
A Path from n_1 to n_k	a sequence of nodes $n_1,n_2,\ldots n_k$ such that n_i is the parent of n_{i+1} for $0<i<k$. The length of this path is the number of edges on the path.
Ancestor and **Descendant**	If there is a path from n_1 to n_k, we say n_k is the descendant of n_1 and n_1 is the ancestor of n_k
Level or **Depth** of node	The length of the unique path from root to this node
Height of a tree	The maximum level of any leaf in the tree



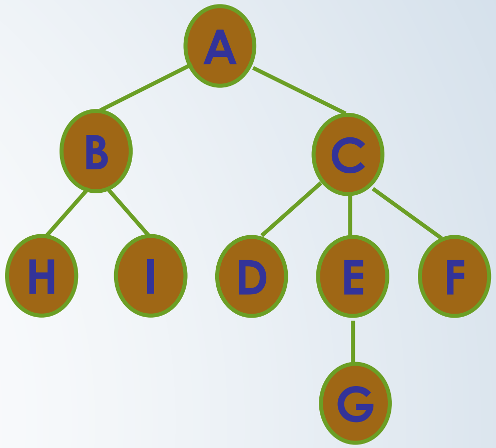
Every node is an ancestor of itself. Every node is an descendent of itself.

A *proper ancestor* of *n* is any node *y* such that node *y* is an ancestor of node *n* and *y* is not the same node as *n*. A *proper descendent* of n is any node y for which n is an ancestor of y and y is not the same node as n.

Definition and Terminology

Level of node : State the levels of all the nodes:

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



Root of a tree: Root of the tree is: _____

Height of a tree: Height of the tree is: _____

Degree of a node : State the degrees of:

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

Terminal node or **leaf**: State all the leaf nodes: ______________________

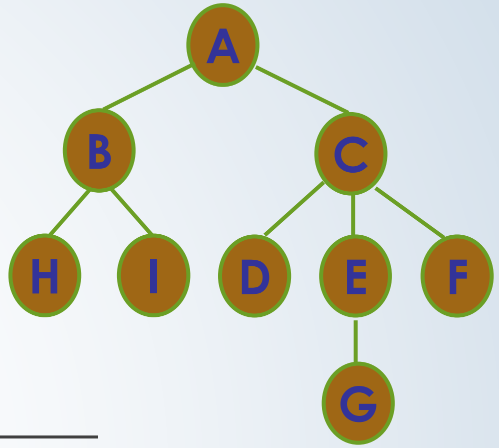
Branch node: State all the branch nodes: ______________________

Definition and Terminology

Parent and **Siblings**:

State the parents of: A:___, B:___, C:___,
D:___, E:___, F:___,
G:___, H:___, I:___

State the siblings of: A:________, B:________,
C:________, D:________, E:________,
F:________, G:________, H:________, I:________



Ancestor and **Descendant**:

State the ancesters of: A:_______, B:_______, C:_______, D:_____________,
E:_____________, F:_____________, G:_____________,
H:_____________, I:_____________

State the descendants of: A:______________________________,
B:_____________, C:_____________,
D:____, E:____, F:____, G:____, H:____, I:____

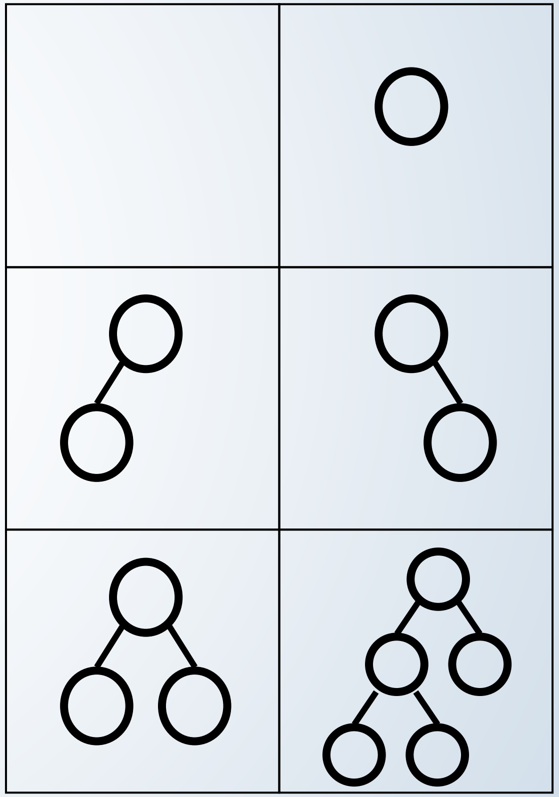
Binary Tree

Definition:

Binary tree can be defined as a finite set of nodes that either

- is empty, or
- consists of
  (1) a root, and
  (2) the elements of 2 disjoint binary trees called the left and right subtrees of the root.

6 Examples of Binary tree:



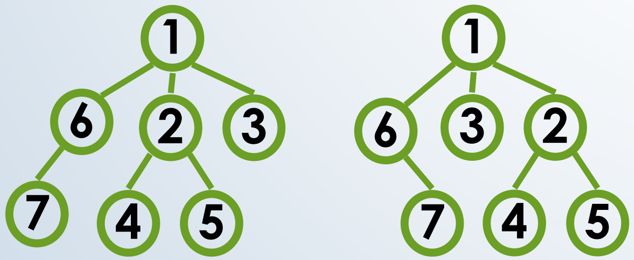
Binary Tree

Comparison:

Tree

- A tree must have at least 1 node
- Each node has 0, 1, 2, .. or many subtrees.
- We don't distinguish subtrees according to their orders.

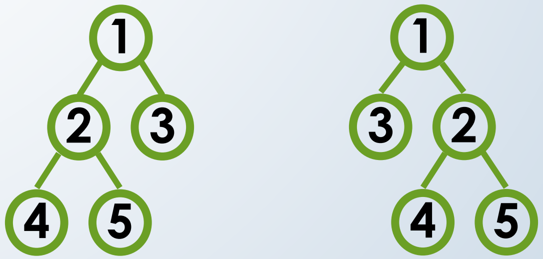
For example, these are __________:



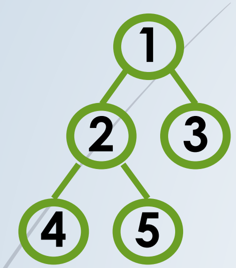
Binary tree

- A binary tree may be empty
- Each node has 0, 1, or 2 subtrees.
- We distinguish between the left and right subtree.

For example, these are ________:



Properties of Binary Tree

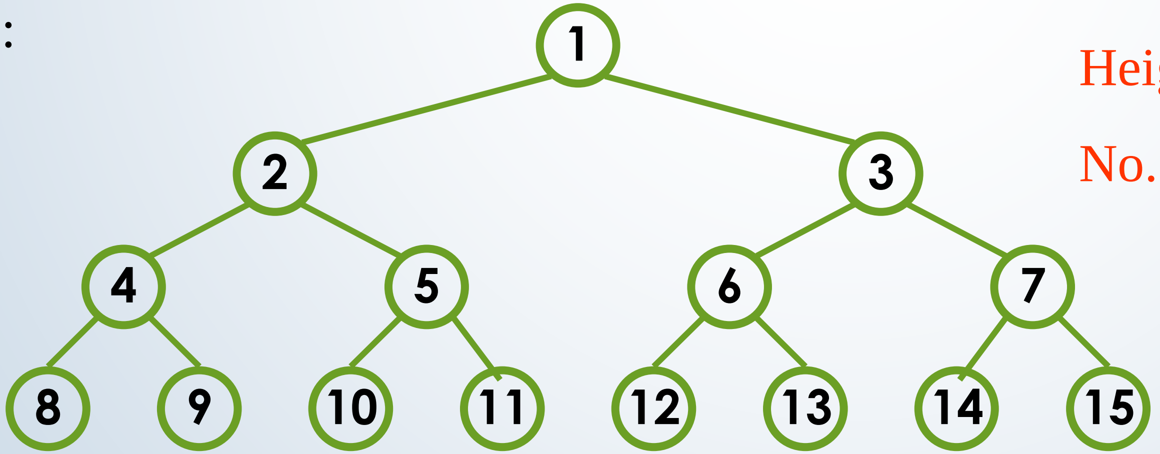


Maximum number of nodes

- Consider the levels of a binary tree: level 1, level 2, level 3, ..
- Maximum number of nodes on a level is 2^{level_id-1}.
- Maximum number of nodes in a binary tree is $2^{height\_of\_tree+1} - 1$.

Full Binary Tree: **No. of nodes = $2^{height\_of\_tree +1}$-1**

Example:



Height of tree = 3

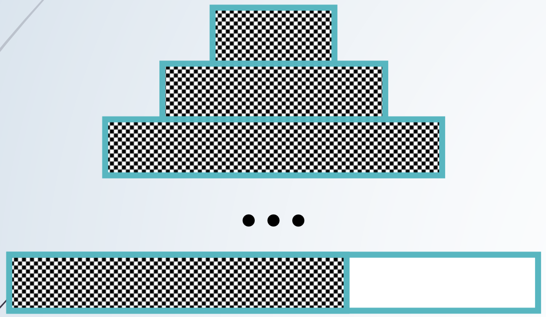
No. of nodes = $2^{height\_of\_tree +1}$-1

= ______

Properties of Binary Tree

Complete Binary Tree:

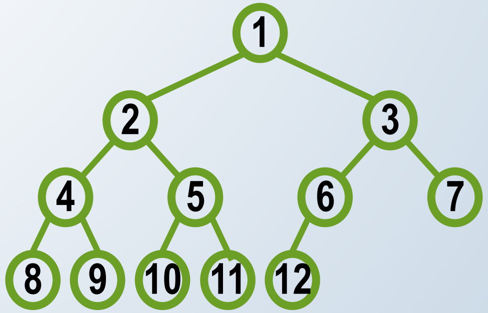
A complete binary tree is like a full binary tree,
But in a complete binary tree,



- Except the bottom level: all are fully filled.
- The bottom level: The filled slots are at the left of the empty slots (if any).

Definition: A binary tree with ***n*** nodes and height ***k*** is **complete** if and only if its nodes correspond to the nodes numbered from 1 to ***n*** in the fully binary tree of height ***k***.

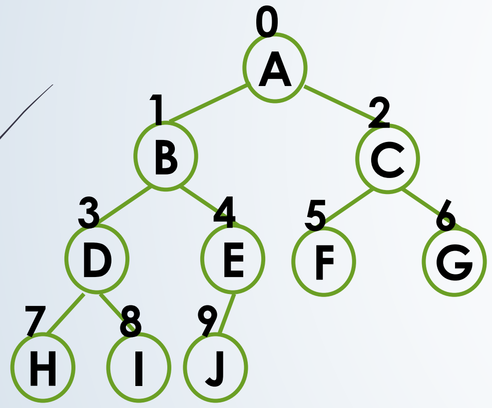
- Each leaf in a tree is either at level ***k*** or level ***k***+1
- Each node has exactly 2 subtrees at level 1 to level ***k***-1



Array Representation of Binary Tree

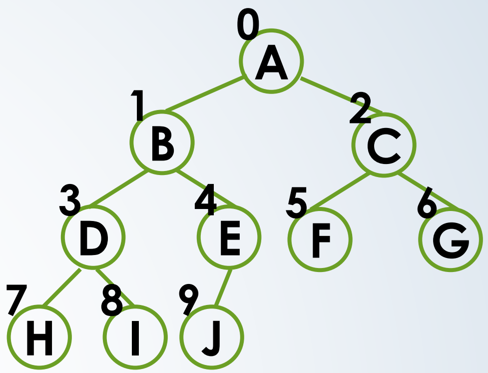
Array Representation of Binary Tree

A numbering scheme:



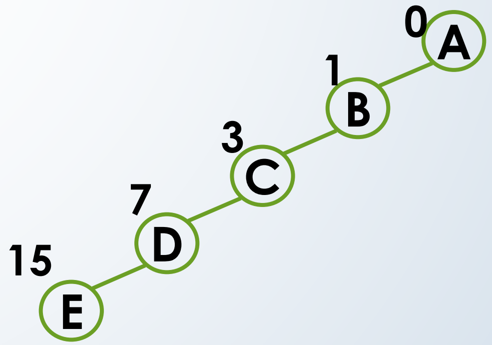
We can **represent binary trees using array** by applying this numbering scheme.

Example 1:



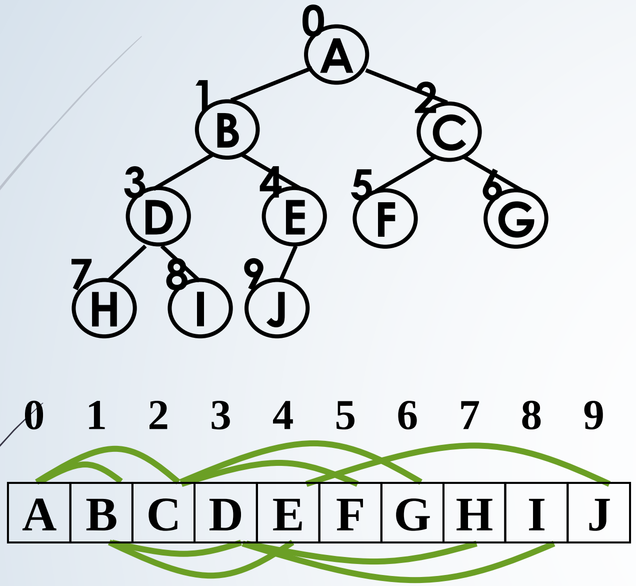
0	1	2	3	4	5	6	7	8	9
A	B	C	D	E	F	G	H	I	J

Example 2:



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Array Representation of Binary Tree



Children of a node at slot ***i***:

Left(i) = 2i+1

Right(i) = 2i+2

Parent of a node at slot ***i***:

Parent(i) = $\lfloor (i-1)/2 \rfloor$

$\lfloor x \rfloor$: **"Floor" The greatest integer less than x**

$\lceil x \rceil$: **"Ceiling" The least integer greater than x**

For any slot ***i***,

If ***i*** is odd: it represents a left son.

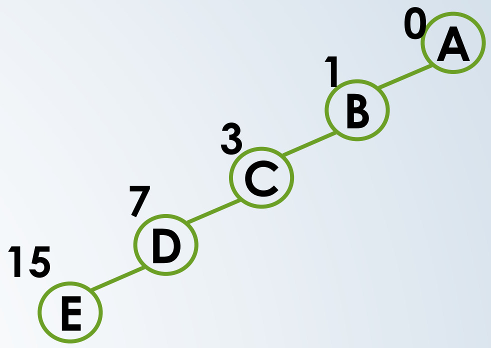
If ***i*** is even (but not zero): it represents a right son.

The node at the right of the represented node of ***i*** (if any), is at ***i***+1.

The node at the left of the represented node of ***i*** (if any), is at ***i***-1.

Array Representation of Binary Tree

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	B	-	C	-	-	-	D	-	-	-	-	-	-	-	E



Unused array elements (not exist or is NULL) must be flagged for non-full binary tree.

Solutions: 1. put a special value in the location

2. Add a "used" field (true/false) to each node.

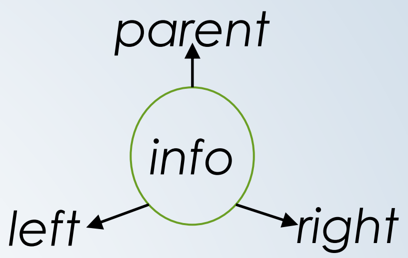
<u>Advantages and Disadvantages of using array to represent binary tree:</u>

- Simpler
- Save storage for trees known to be almost full.
- Waste of space (except complete binary tree)
- Maximum size of the tree is fixed in advance
- Inadequacy: insertion and deletion of nodes from the middle of a tree require the movement of potentially many nodes

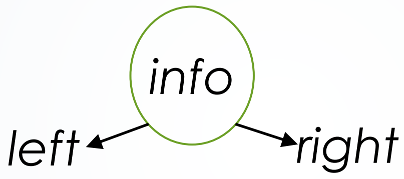
Linked Representation of Binary Tree

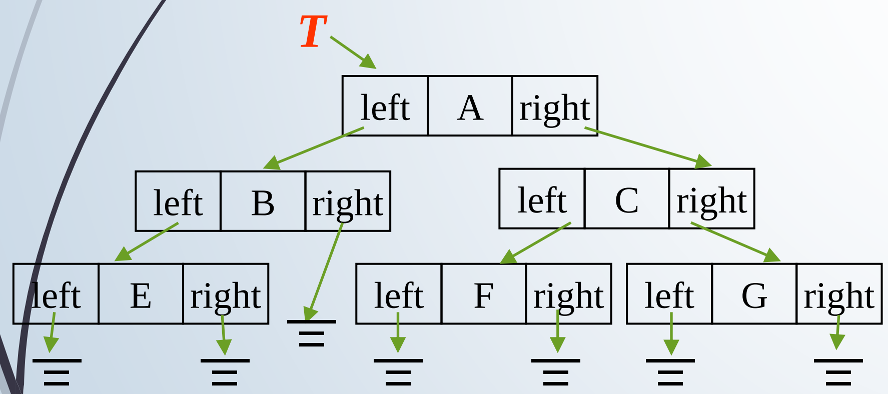
Link Representation of Binary Tree

- Each node can contains *info*, *left*, *right*, *parent* fields
- where *left*, *right*, *parent* fields are node pointers pointing to the node's left son, right son, and parent, respectively.



- If the tree is always traversed in downward fashion (from root to leaves), the parent field is unnecessary.





T.left(p): Return the position that represents the left child of p, or None if p has no left child.
T.right(p): Return the position that represents the right child of p, or None if p has no right child.
T.sibling(p): Return the position that represents the sibling of p, or None if p has no sibling.

- If the tree is empty, root = NULL; otherwise from root you can find all nodes.
- root->left and root->right point to the left and right subtrees of the root, respectively.

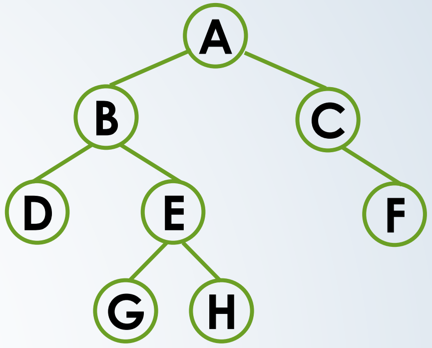
Binary Tree Operations - height

Review:

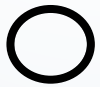
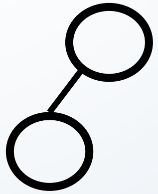
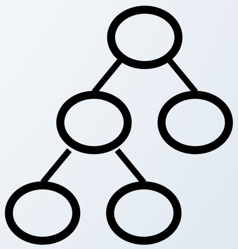
Depth of node : The depth of root(T) is zero.

The depth of any other node is one larger than his parent's depth

Height of a tree: The maximum depth of any leaf in the tree

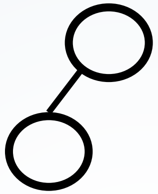
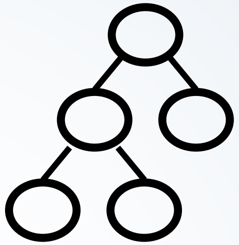


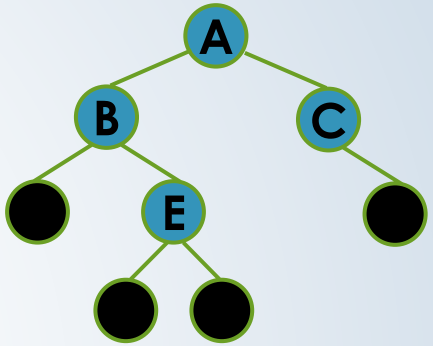
Example:

Height of a NULL binary tree is 0	 Height of a tree with 1 node is 0	 Height = 1	 Height = 2

Binary Tree Operations - countleaves

Example:

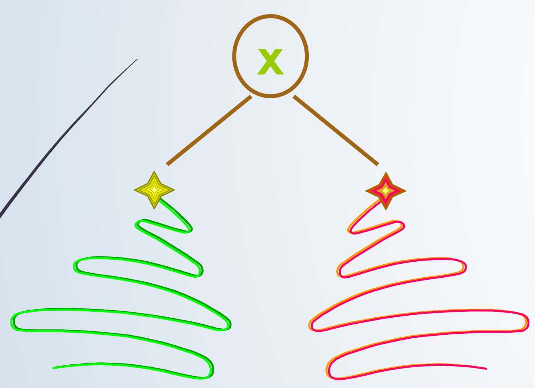
A NULL binary tree has 0 leaf node	 **A tree with 1 node has 1 leaf node**	 **No. of leaf nodes = 1**	 **No. of leaf nodes = 3**

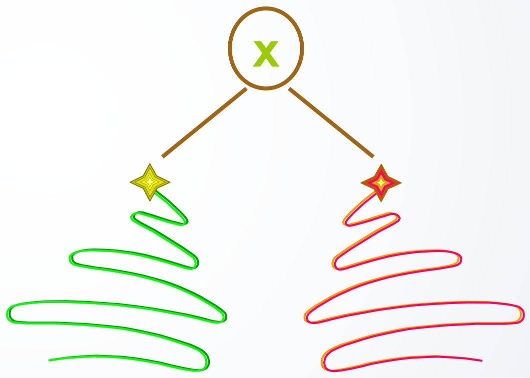



```
//To count the number of leaf nodes

def count_leaf(p):
    if p == None):
        return 0
    elif ((p.left == None) and (p.right == None)):
        return 1
    else:
        return(count_leaf(p.left) + count_leaf(p.right))
```

Binary Tree Operations - equal

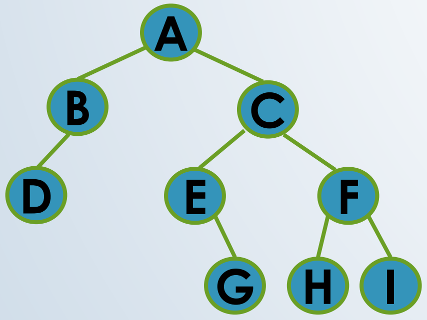




Traversing Binary Tree

Traversing / walking through

A method of examining the nodes of the tree systematically so that each node is visited exactly once.



Three principle ways:

When the binary tree is empty, it is "traversed" by doing nothing, otherwise:

preorder traversal

1. Visit the root
2. Traverse the left subtree
3. Traverse the right subtree

A B D C E G F H I

inorder traversal

1. Traverse the left subtree
2. Visit the root
3. Traverse the right subtree

D B A E G C H F I

postorder traversal

1. Traverse the left subtree
2. Traverse the right subtree
3. Visit the root

D B G E H I F C A

Preorder Traversal

- The order of visitation of nodes is "root, left, right"
  - first visit the node at the root of any subtree
  - then visit its left child
  - then visit its right child
- Any child may itself be the root of a subtree, so this traversal is inherently recursive.

```
procedure PREORDER(T)
visit T
if there is a left child, PREORDER(left child(T))
if there is a right child, PREORDER(right child(T))
```

Inorder Traversal

- The order of visitation of nodes is "left, root, right"
  - first visit its left child
  - then visit the node at the root of any subtree
  - then visit its right child
- Any child may itself be the root of a subtree, so this traversal is also inherently recursive.

```
procedure INORDER(T)
if there is a left child, INORDER(left child(T))
visit T
if there is a right child, INORDER(right child(T))
```

Postorder Traversal

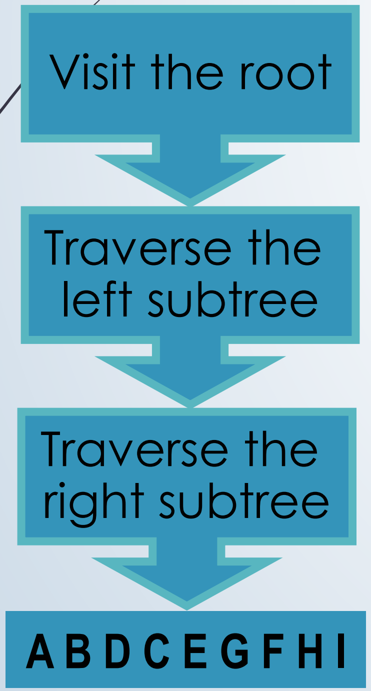
- The order of visitation of nodes is “left, right, root”
  - first visit its left child
  - then visit its right child
  - then visit the node at the root of any subtree
- Any child may itself be the root of a subtree, so this traversal is also inherently recursive.

```
procedure POSTORDER(T)
if there is a left child, POSTORDER(left child(T))
if there is a right child, POSTORDER(right child(T))
visit T
```

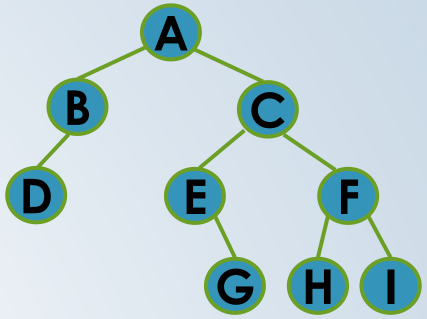
Traversing Binary Tree

When the binary tree is empty, it is "traversed" by doing nothing, otherwise:

preorder traversal



Example:



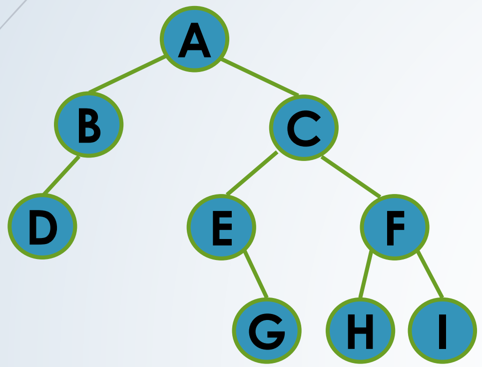
Result:

= A (A's left) (A's right)
= A B **(B's left)** (B's right = NULL) (A's right)
= A B **(B's left)** (A's right)
= A B D (D's left=NULL) (D's right = NULL) (A's right)
= A B D (A's right)
= A B D C **(C's left)** (C's right)
= A B D C **E (E's left=NULL) (E's right)** (C's right)
= A B D C **E (E's right)** (C's right)
= A B D C **E G (G's left=NULL) (G's right = NULL)** (C's right)
= A B D C **E G** (C's right)
= A B D C **E G** F (F's left) (F's right)
= A B D C **E G** F H (H's left=NULL) (H's right =NULL) (F's right)
= A B D C **E G** F H I (I's left=NULL) (I's right =NULL)
= A B D C **E G** F H I

Traversing Binary Tree

Exercise:

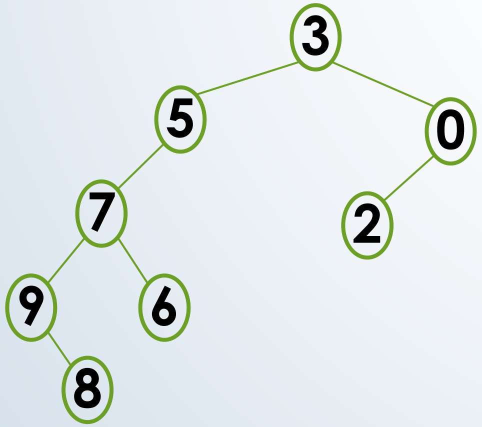
1. Examine the inorder and postorder traversals of the tree:



inorder:

postorder:

2. Examine the preorder, inorder and postorder traversals of the tree:



preorder:

inorder:

postorder:

Traversing Binary Tree

Reconstruction of Binary Tree from its preorder and Inorder sequences

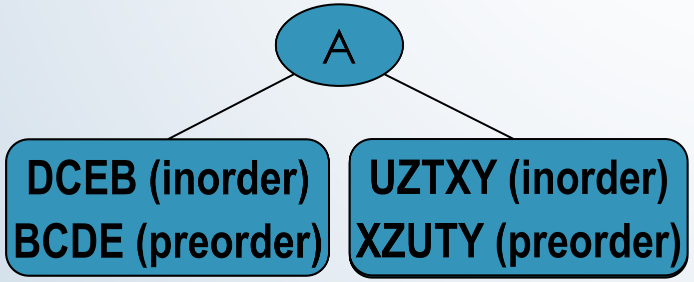
Example: Given the following sequences, find the corresponding binary tree:

preorder : ABCDEXZUTY
inorder : DCEBAUZTXY

Looking at the whole tree:

- "preorder : **A**BCDEXZUTY"
  ==> A is the root.
- Then, "inorder : DCEB**A**UZTXY"

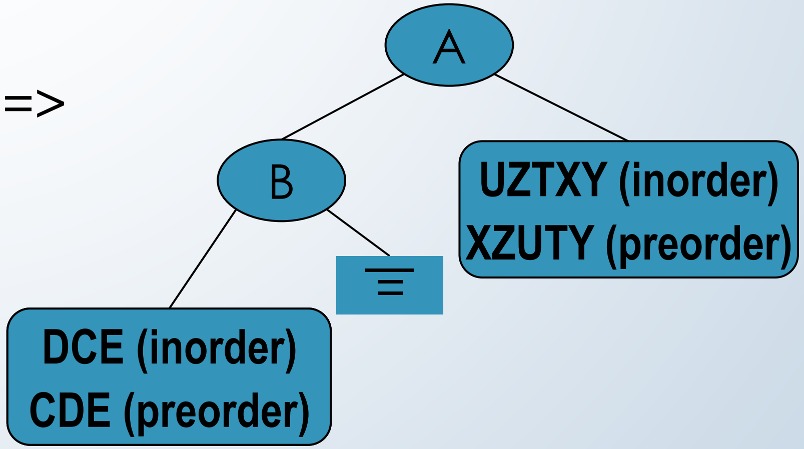
==>



Looking at the left subtree of A:

- "preorder : BCDE"
  ==> B is the root
- Then, "inorder: DCE**B**"

=>



Traversing Binary Tree

Reconstruction of Binary Tree from its preorder and Inorder sequences

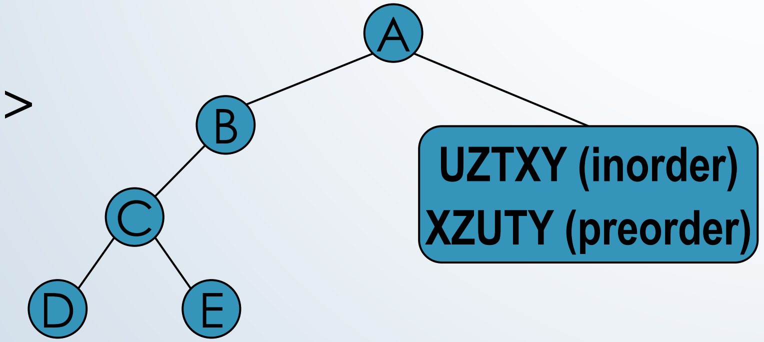
Example: Given the following sequences, find the corresponding binary tree:

preorder : ABCDEXZUTY
inorder : DCEBAUZTXY

Looking at the left subtree of B:

- "preorder : CDE"
  ==> C is the root
- Then, "inorder: D**C**E"

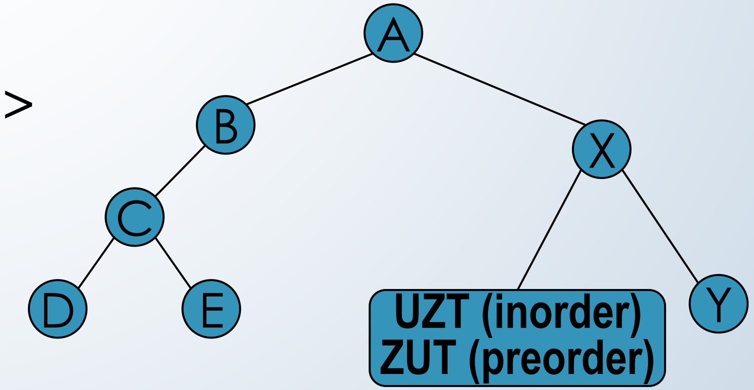
=>



Looking at the right subtree of A:

- "preorder : XZUTY"
  ==> X is the root
- Then, "inorder: UZT**X**Y"

=>



Traversing Binary Tree

Reconstruction of Binary Tree from its preorder and Inorder sequences

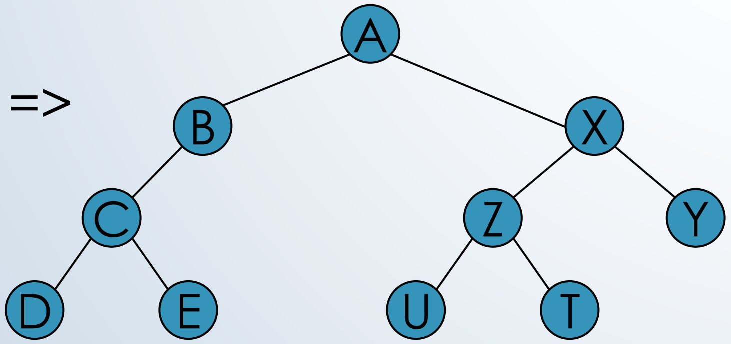
Example: Given the following sequences, find the corresponding binary tree:

preorder : ABCDEXZUTY
inorder : DCEBAUZTXY

Looking at the left subtree of X:

- "preorder : ZUT"
  ==> Z is the root
- Then, "inorder: U**Z**T"

=>



Traversing Binary Tree

But: A binary tree may not be uniquely defined by its preorder and postorder sequences.

Example: Preorder sequence: ABC
Postorder sequence: CBA

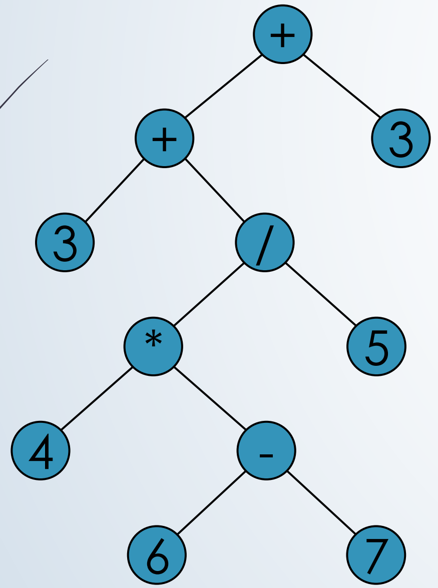
We can construct 2 different binary trees:

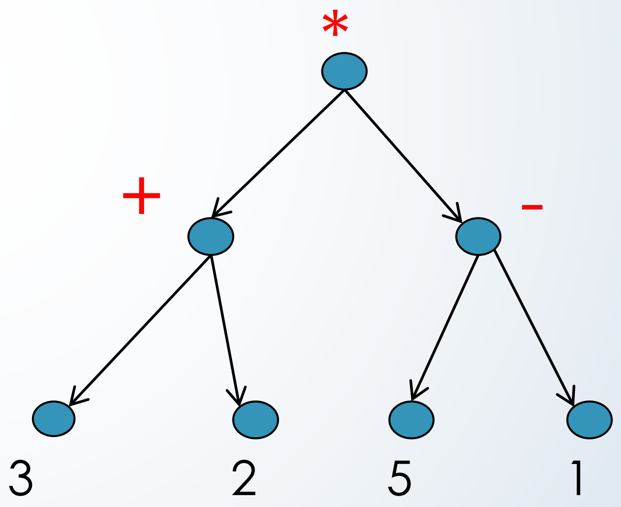
Applications of Binary Tree

Representation of General Function Expression

Example:

For **3+4*(6-7)/5+3**:



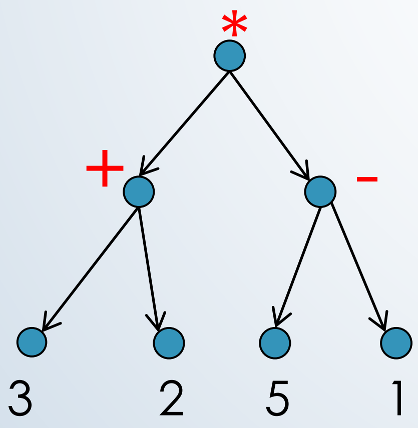


Level Traversal

- The order that nodes are visited is based on their distance from the root node.
  - first the root node is visited
  - then all those nodes that are of distance 1 to the root are visited
  - and then those nodes that are of distance 2 to the root are visited
  - ...
- Since the standard binary representation of a tree does not allow for direct determination of all nodes on the same level, a queue must be used to maintain that information. By adding the children of the node being visited to the end of the queue, each level will be traversed before going on to the next.

Algorithm for Level Traversal

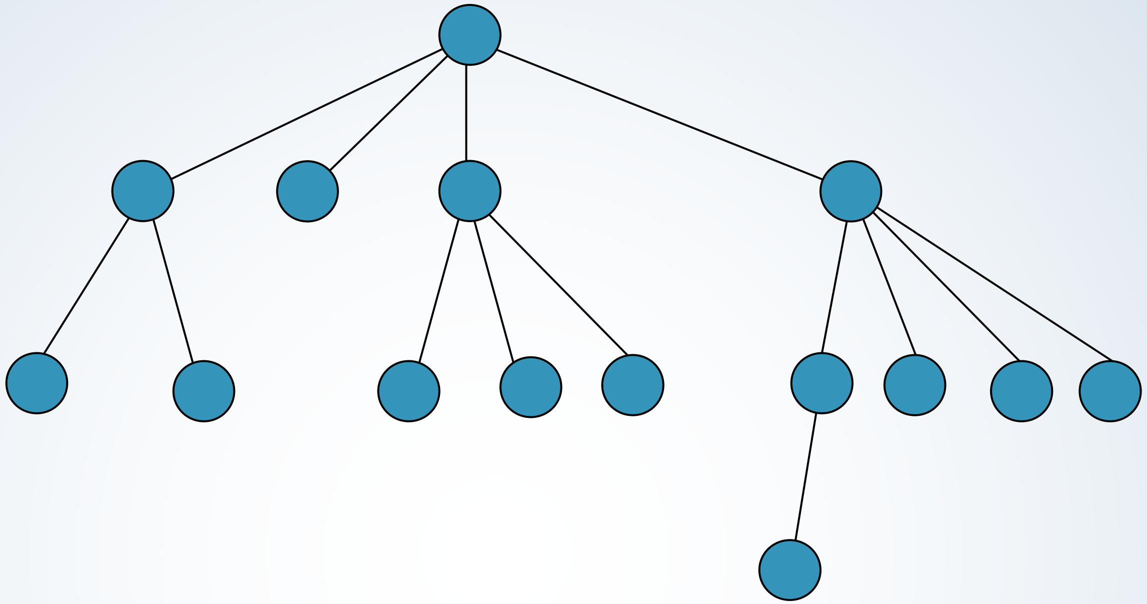
```
add the root node to the queue
while the queue is not empty
    remove a node, T, from the queue
    visit T
    add T's children (if any) to the queue
```



Visiting Order is: * + - 3 2 5 1

Level traversal is not normally used with expression tree. But it is very important when you deal with graphs.

Binary Representation of General Tree



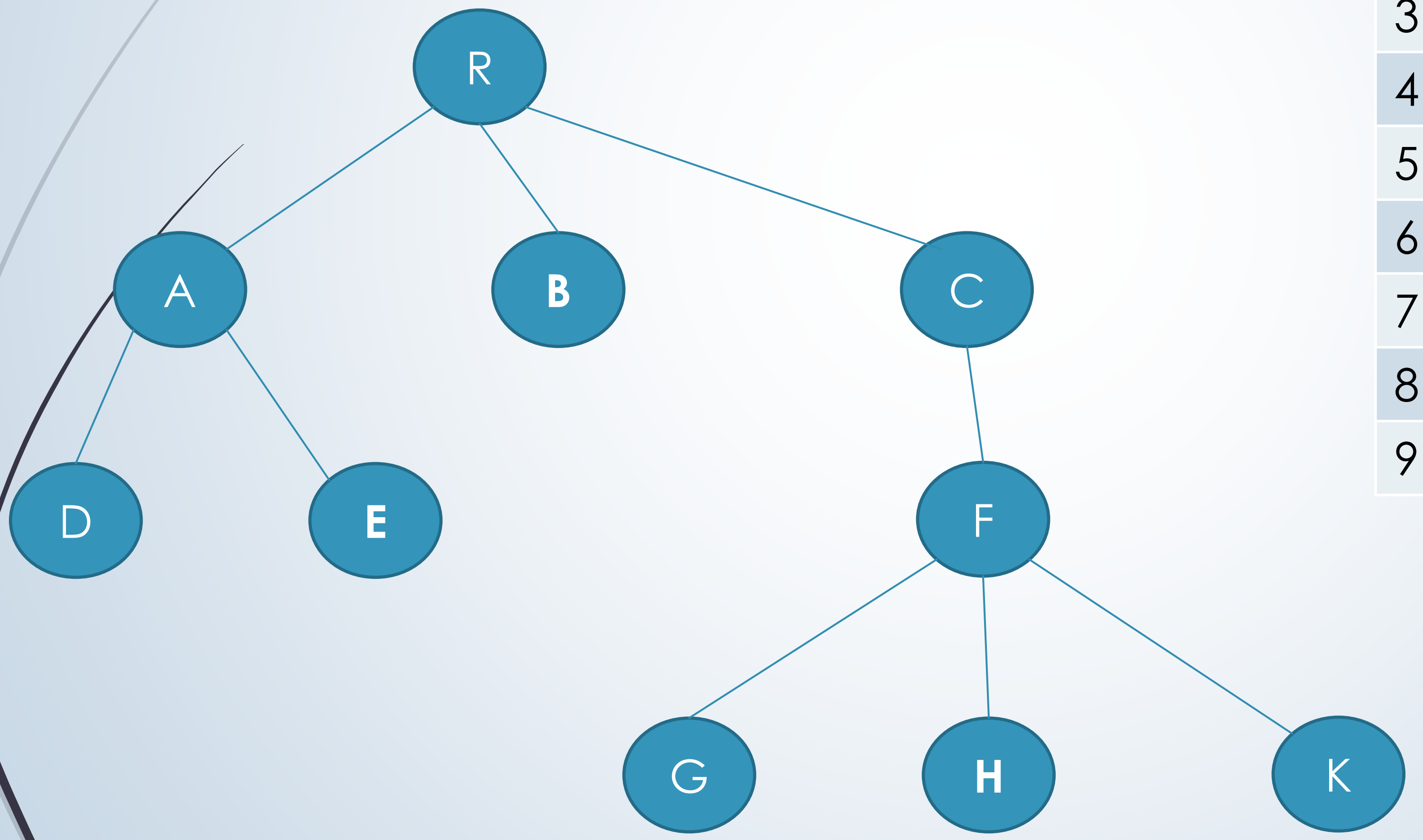
- One node can have many children nodes
- Impossible to make so many links
- Is there a way that each node uses only two links?
  - Link1:
  - Link2:

Tree Representation

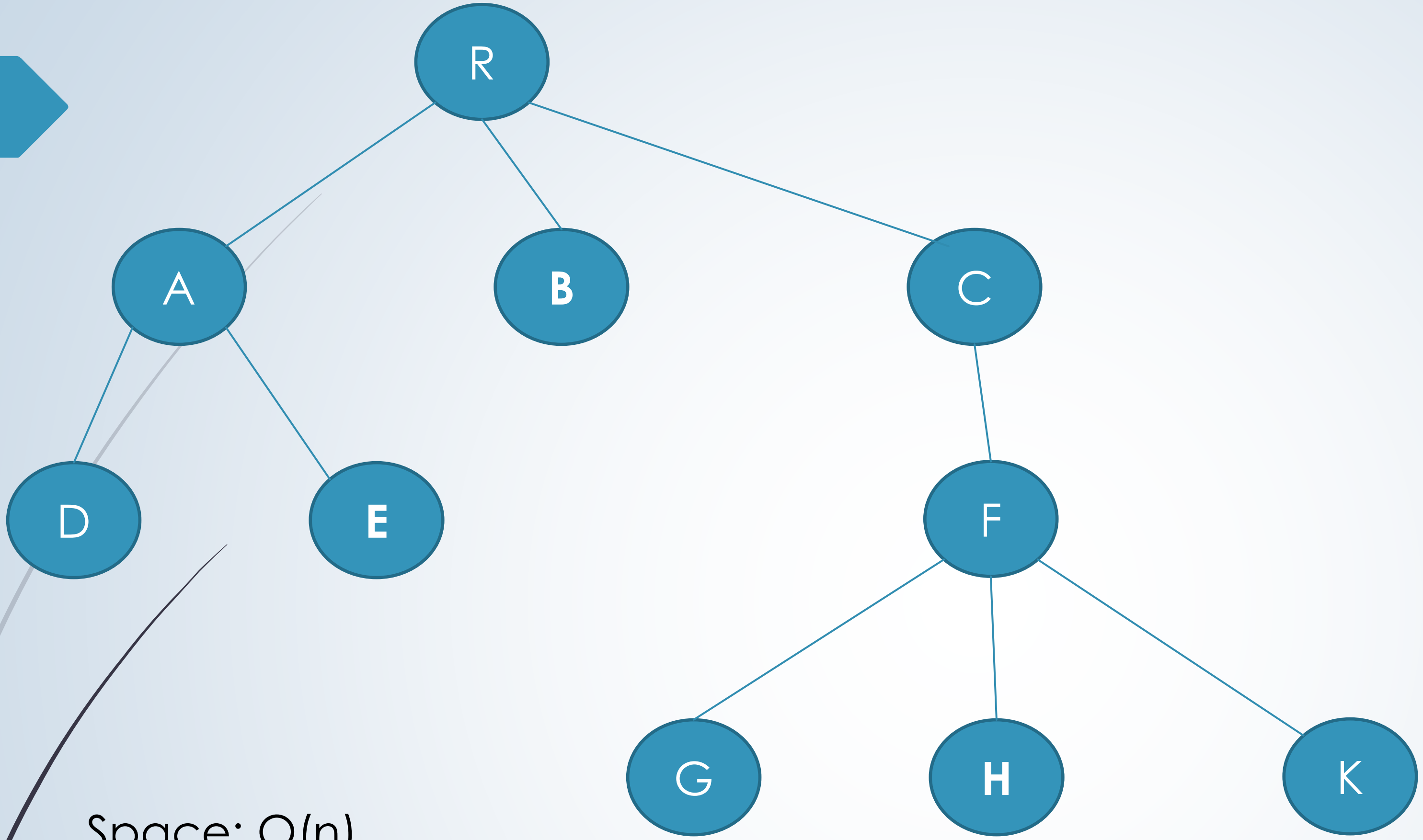
- Interface for each node
  - root()
  - parent()
  - firstChild()
  - nextSibling()
  - insert(i,e): insert e as the i-th child
  - remove(i): remove the i-th child
  - traverse()

Each non-root node has one and only one parent.

Idea: organize all the nodes as a sequence.



rank	data	parent
0	R	-1
1	A	0
2	B	0
3	C	0
4	D	1
5	E	1
6	F	3
7	G	6
8	H	6
9	K	6



rank	data	parent
0	R	-1
1	A	0
2	B	0
3	C	0
4	D	1
5	E	1
6	F	3
7	G	6
8	H	6
9	K	6

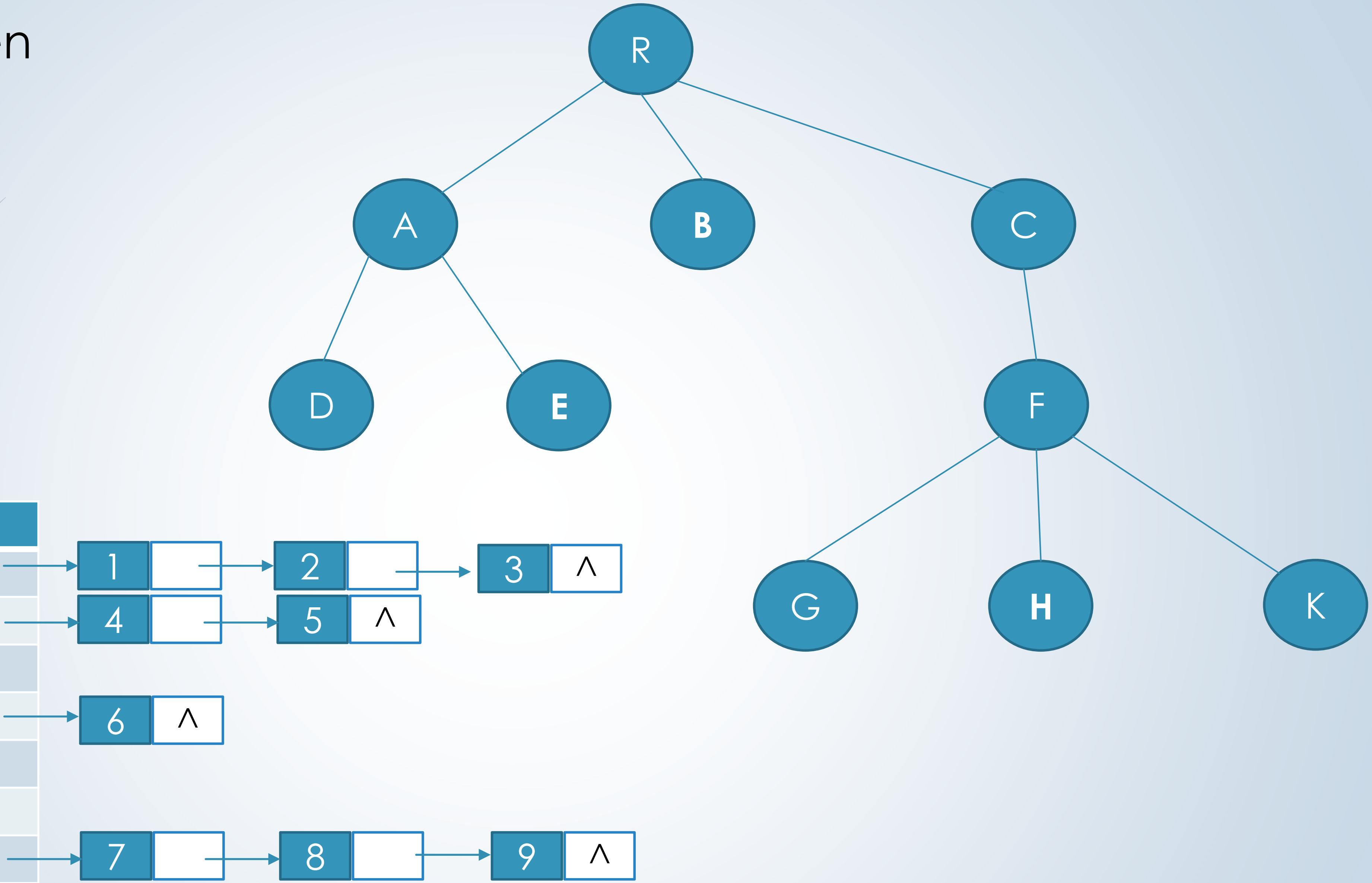
Space: O(n)
Time:

- parent(): O(1)
- root(): O(n) or O(1)
- firstChild(): O(n)
- nextSibling(): O(n)

How to find child or sibling quickly?

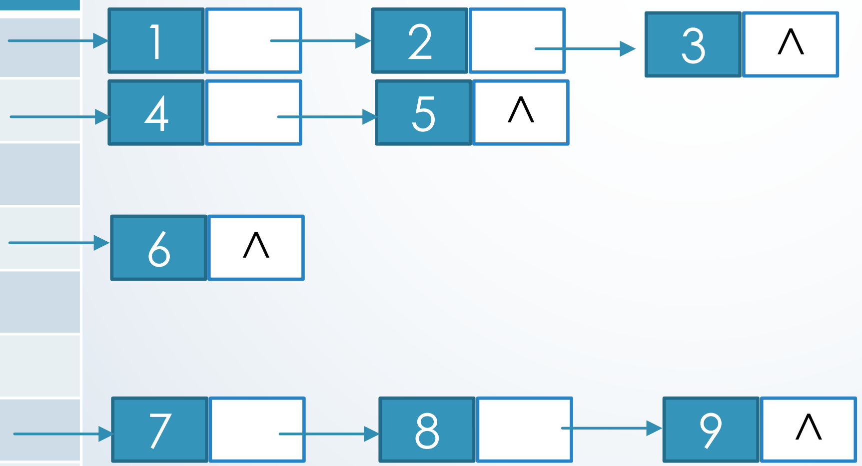
Finding Children

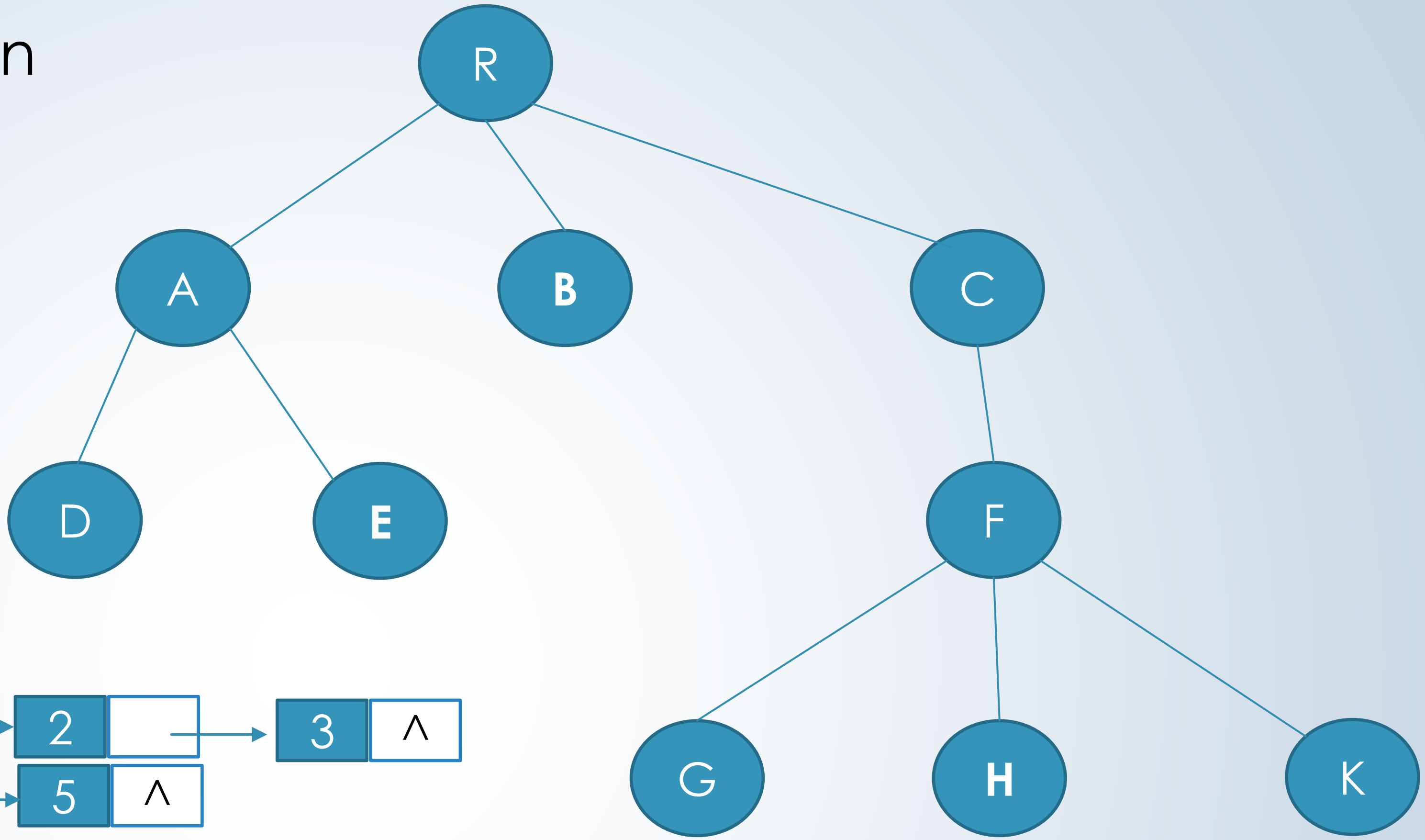
rank	data	children
0	R	
1	A	
2	B	^
3	C	
4	D	^
5	E	^
6	F	
7	G	^
8	H	^
9	K	^



Finding Parent andChildren

rank	data	parent	children
0	R	-1	
1	A	0	
2	B	0	^
3	C	0	
4	D	1	^
5	E	1	^
6	F	3	
7	G	6	^
8	H	6	^
9	K	6	^

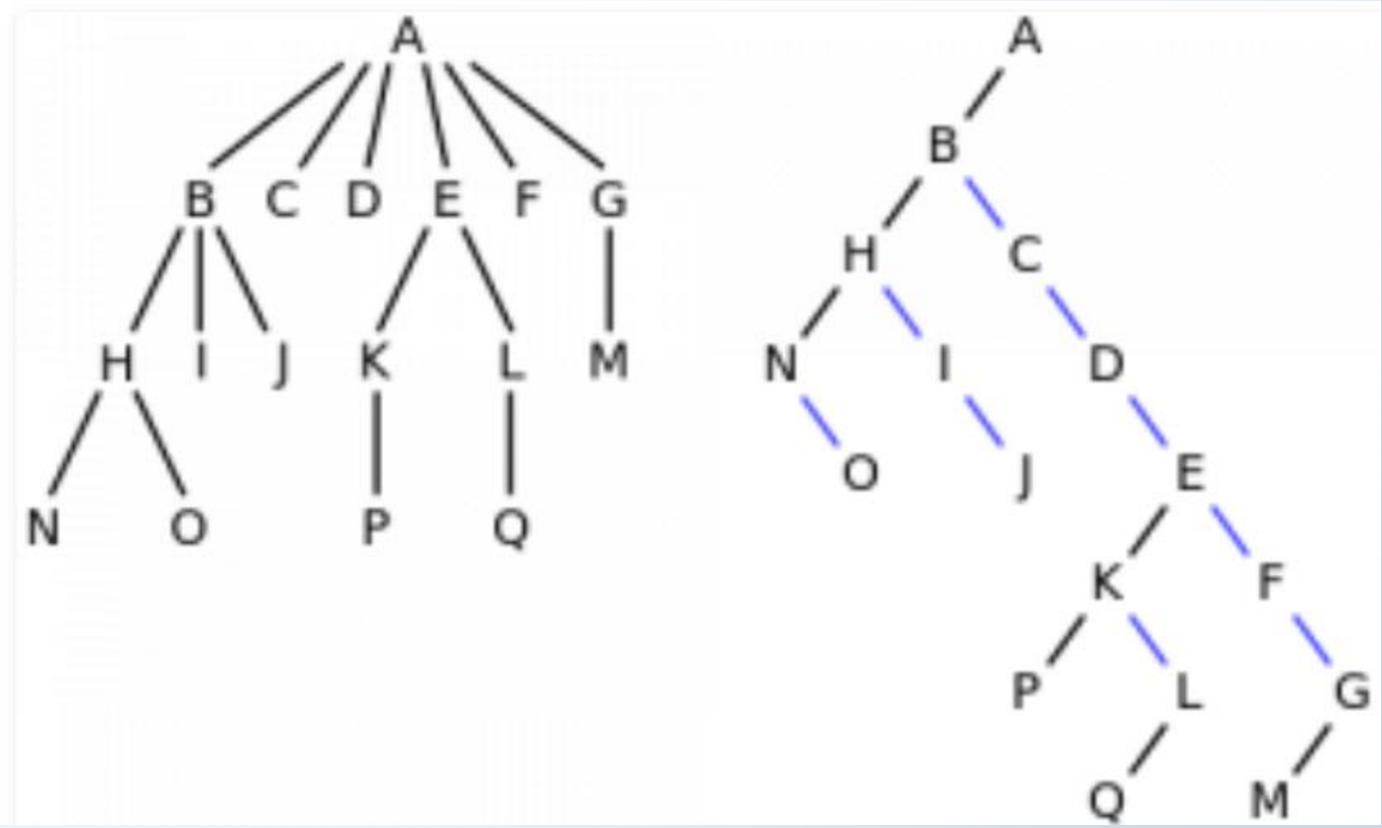




Problem: the degree may vary.

Left-child right-sibling representation

- In an n-ary tree, a node holds just two references, first a reference to its first child, and the other to its immediate next sibling.
- At each node,
  - link children of same parent from left to right.
  - Parent should be linked with only first child.



[Left-Child Right-Sibling Representation of Tree - GeeksforGeeks]