Chapter 5: Tree

Part I

Department of Computer Science and Engineering
Kathmandu University

Contents

Tree: Concept and definition

Non-linear data structure

Each element can have **more than one successor**

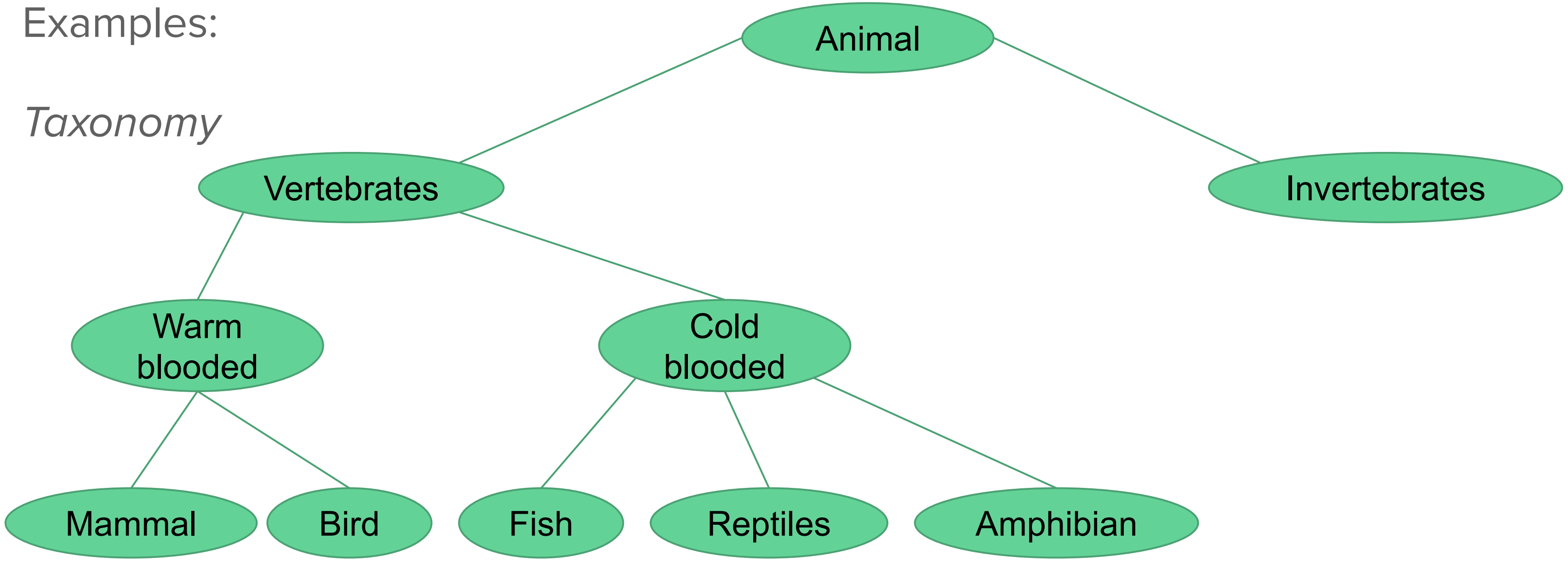
Examples: Trees and graphs

Tree data structure

In tree data structure, **data are organized in a hierarchical manner**.

Examples:

Taxonomy

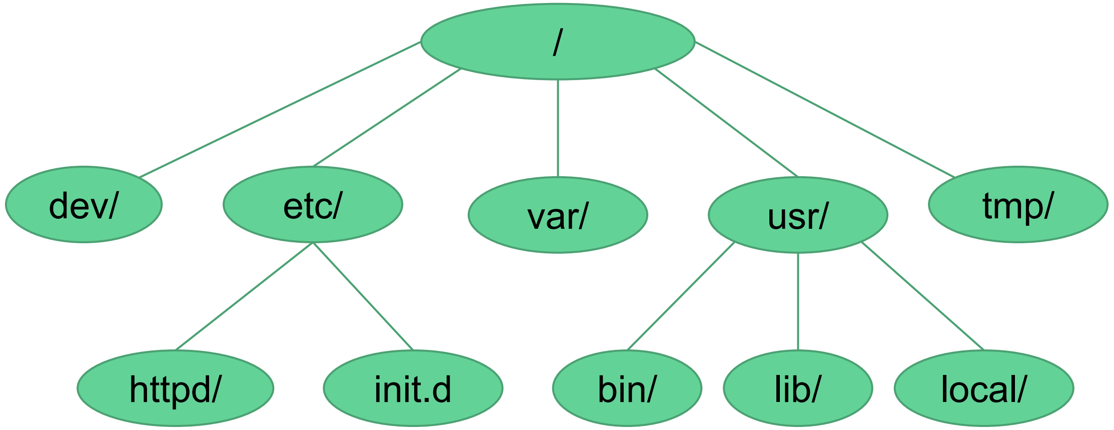


Tree data structure

In tree data structure, **data are organized in a hierarchical manner**.

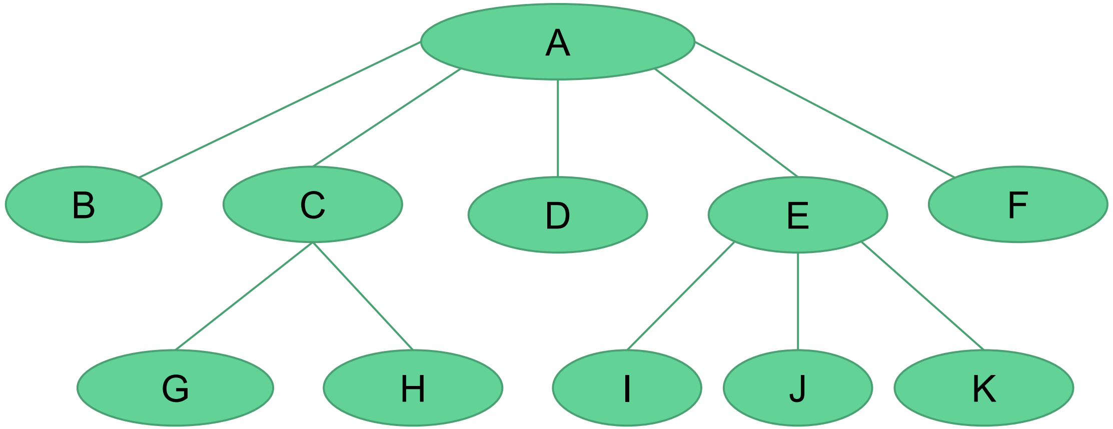
Examples:

File system



Tree data structure

- Consists of a finite set of elements, called **nodes**, and a finite set of directed lines, called **branches**, that connect the nodes.
- Each element/node, except the root node, can have only one predecessor.
- Has no cycles



Tree data structure

Recursive definition:

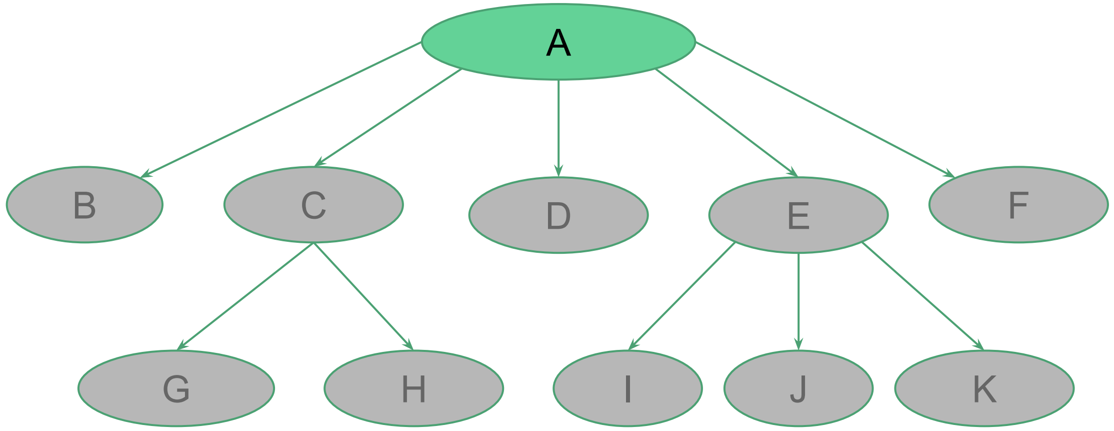
A tree is a finite set of one or more nodes such that

1. There is a specifically designed node called the **root**
2. The remaining nodes are partitioned into $n \geq 0$ **disjoint sets** T_1, T_2, ... , T_n, where each of these sets is a tree.
   T_1, T_2, ..., T_n are called the **subtrees** of the root.

Terminologies

Terminologies

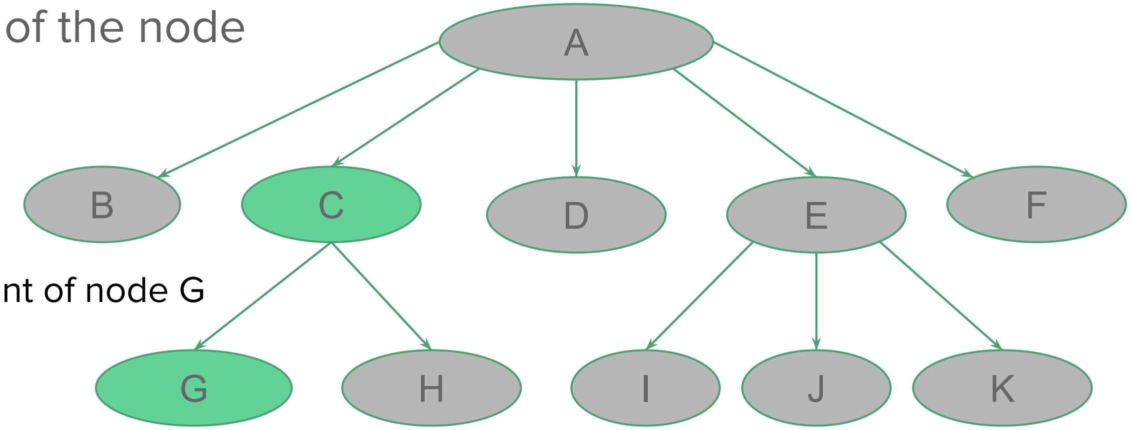
Root - The first/topmost node; has no parent



Terminologies

Parent of a node - Predecessor of the node

Children of a node - Successors of the node

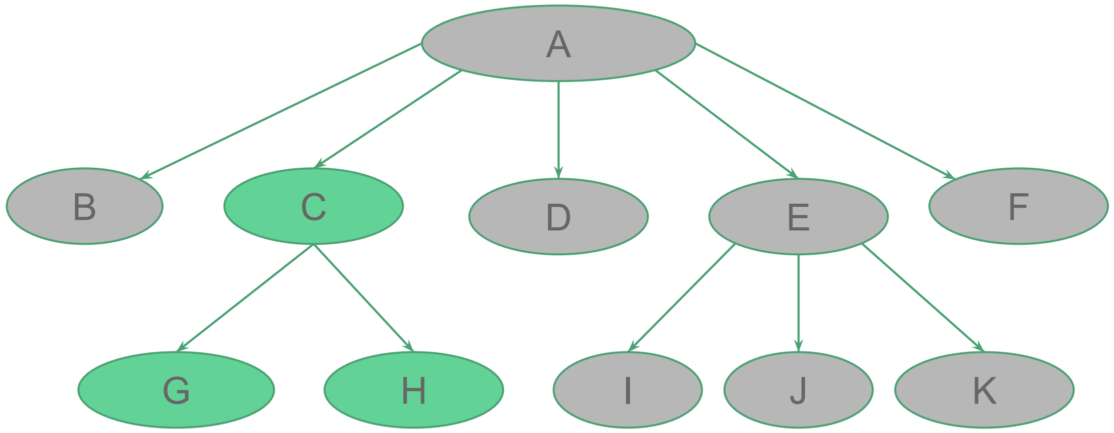


C is the parent of node G

G is a child of node C

Terminologies

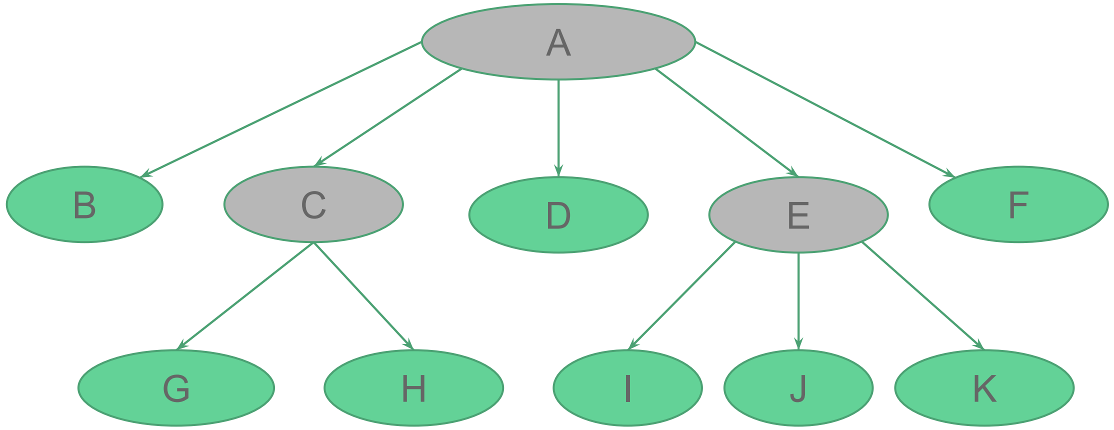
Siblings - Nodes with the same parent



G and H are siblings

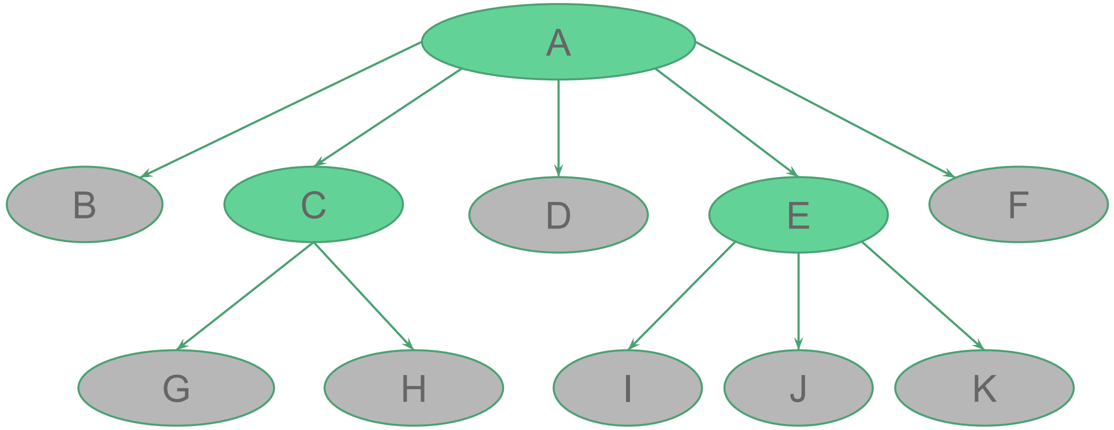
Terminologies

Leaves - Nodes with no children


$$\begin{tikzcd}
 & & & A \arrow[dll] \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr] & \\
B & C \arrow[d] \arrow[dr] & & D & E \arrow[d] \arrow[dr] \arrow[drr] & F \\
 & G & H & I & J & K
\end{tikzcd}$$

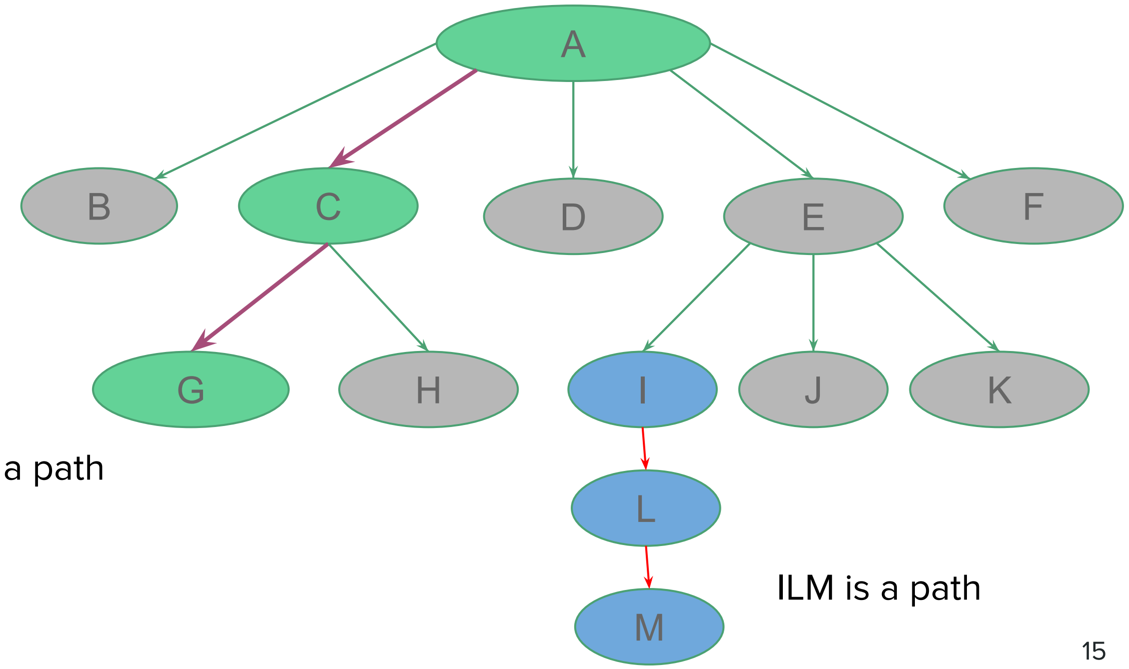
Terminologies

Internal nodes - Nodes with at least one child



Terminologies

Path - An ordered list of nodes that are connected by edges

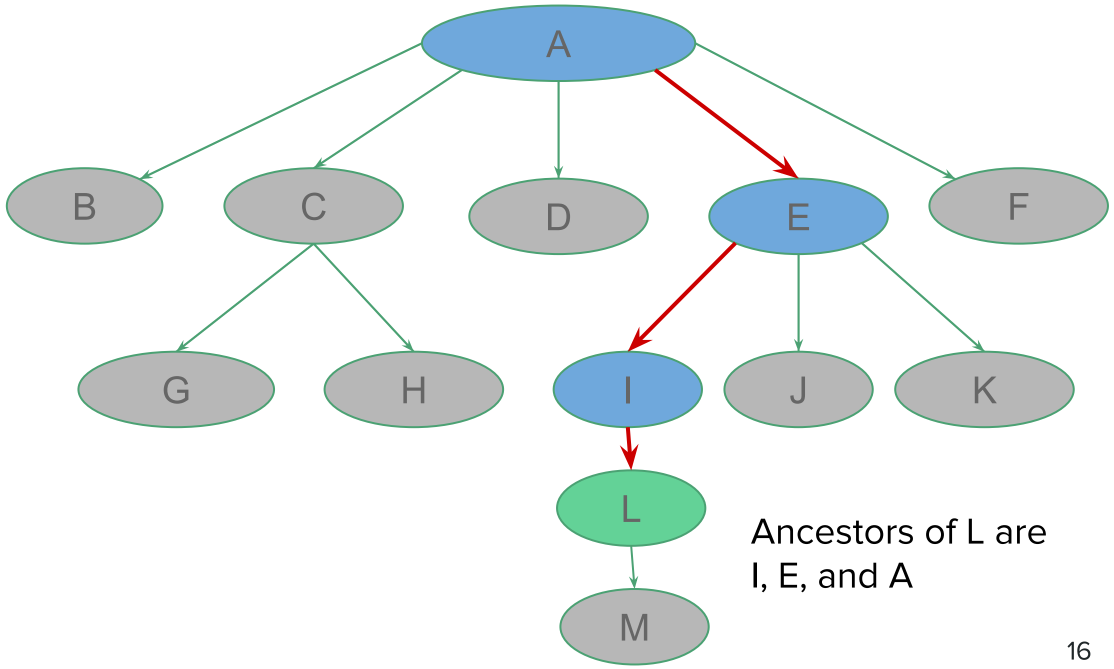


ACG is a path

ILM is a path

Terminologies

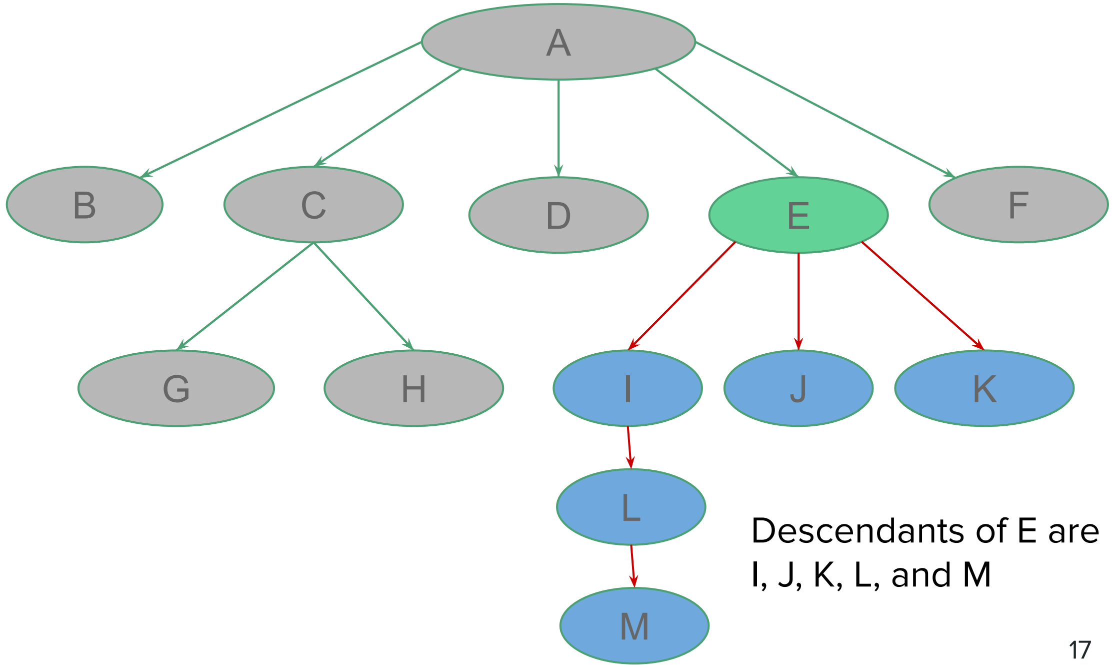
Ancestors of a node - All the nodes along the path from the root to that node



Ancestors of L are I, E, and A

Terminologies

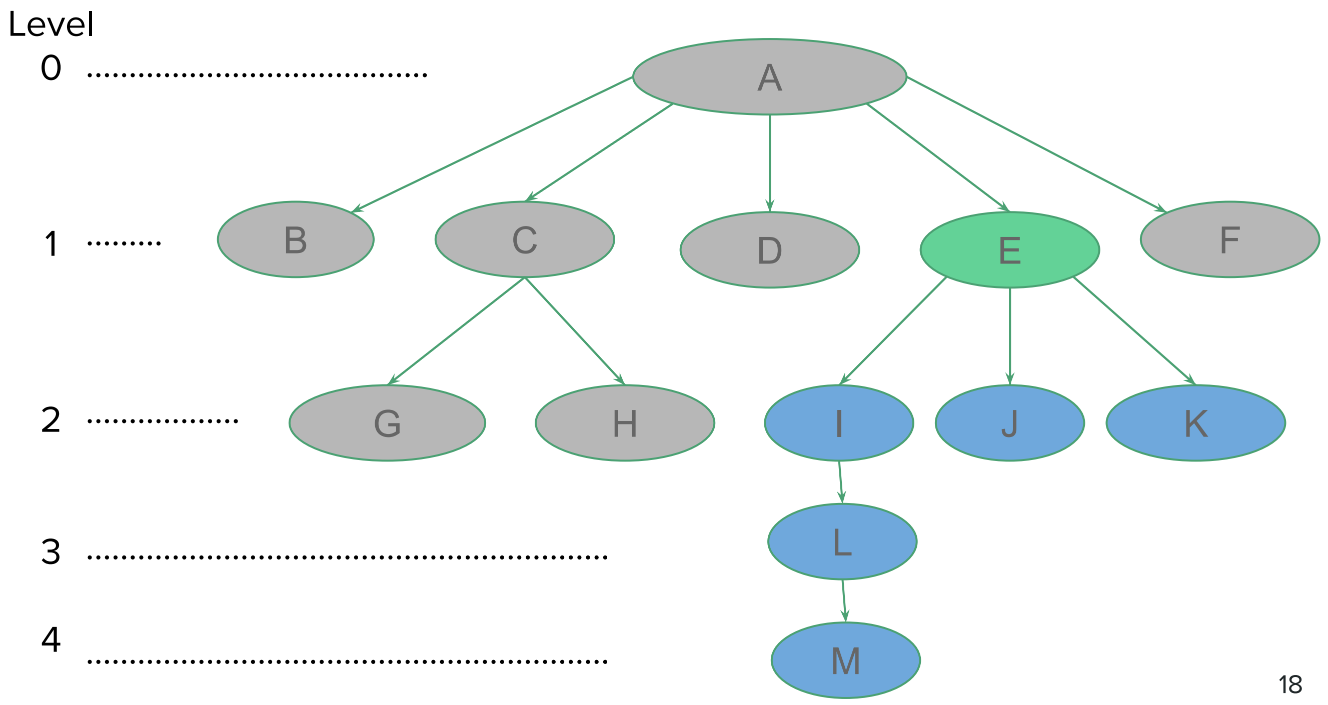
Descendants of a node - All the nodes below that node along the path from that node to the leaves



Descendants of E are I, J, K, L, and M

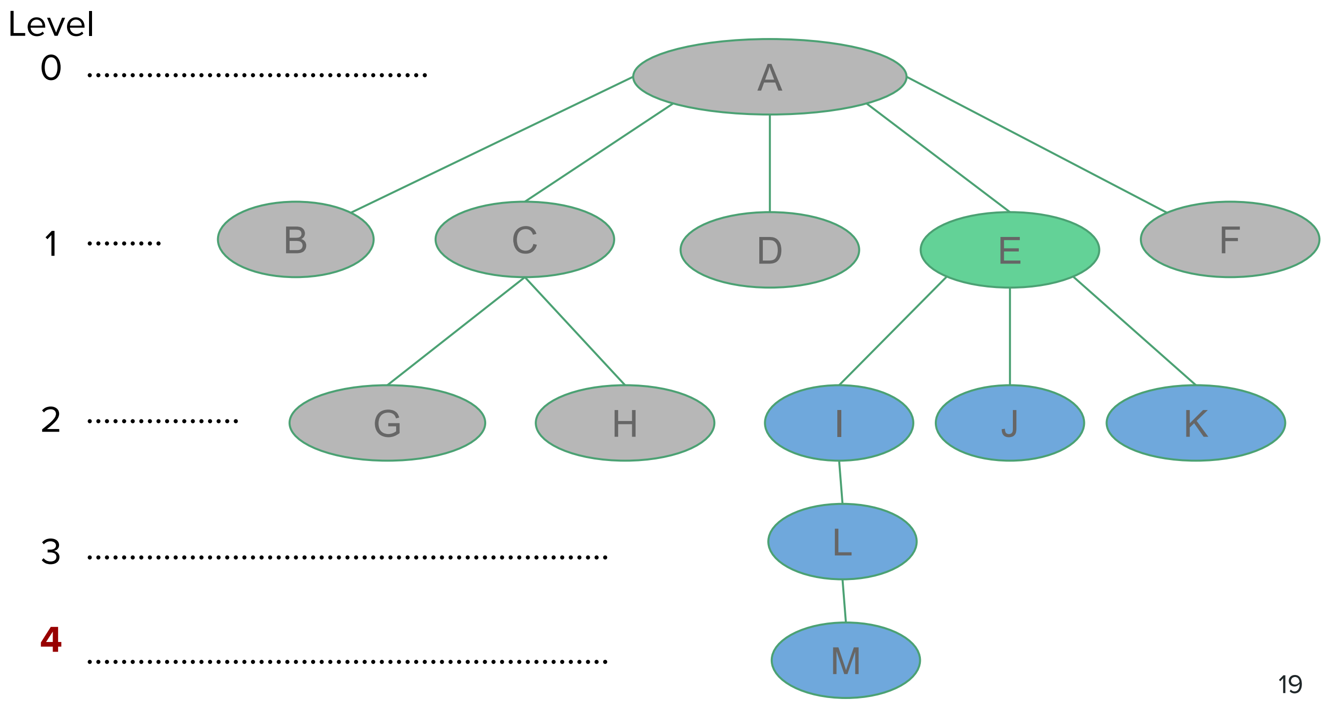
Terminologies

Level of a node - Distance of the node from the root



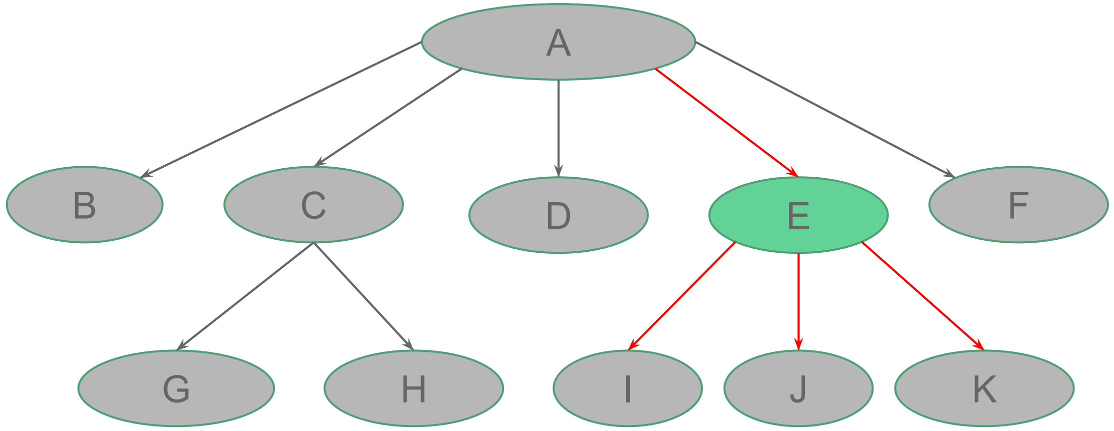
Terminologies

Height of a node - Maximum level of any leaf



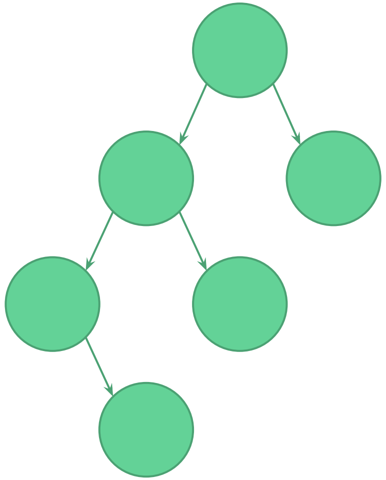
Terminologies

Degree of a node - Number of subtrees of that node



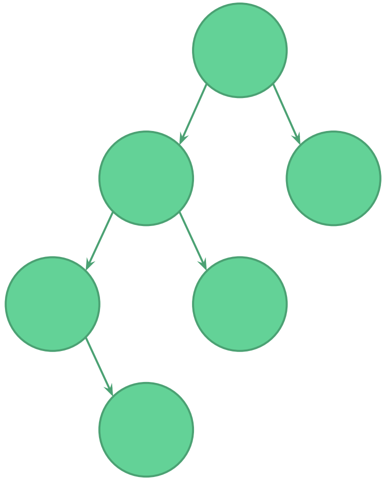
Degree of E = 3
Degree of A = 5

Is it a tree?

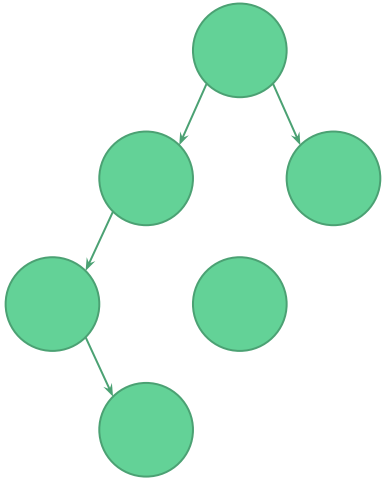


?

Is it a tree?

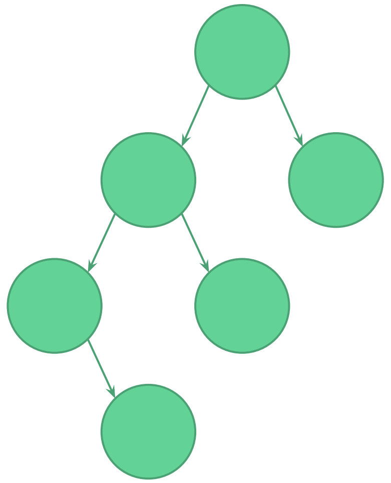


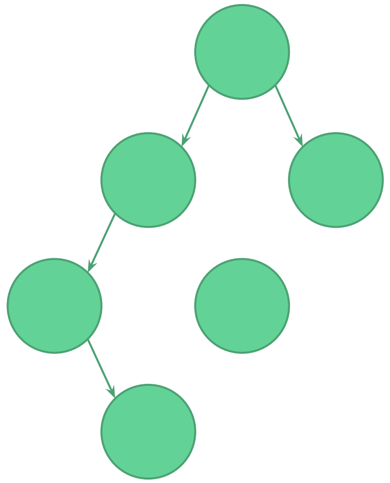
YES

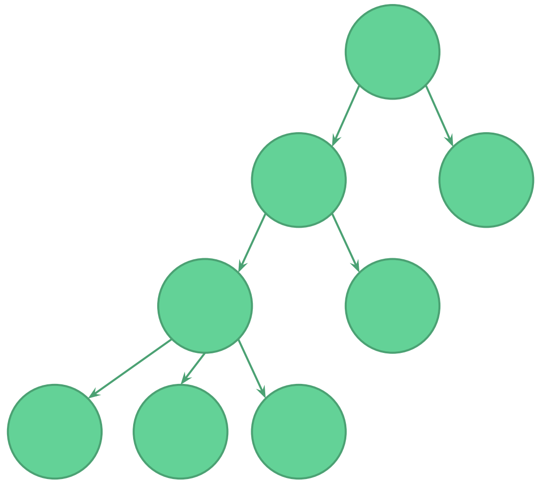


?

Is it a tree?





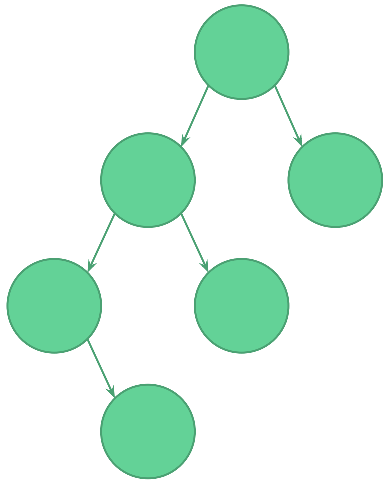


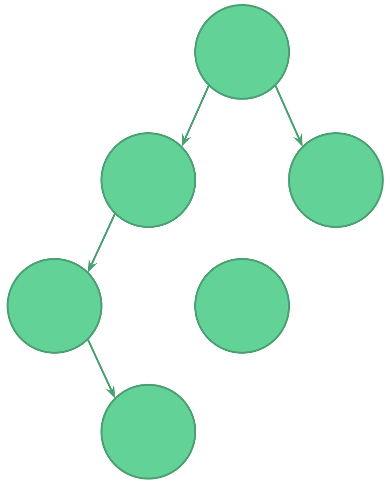
YES

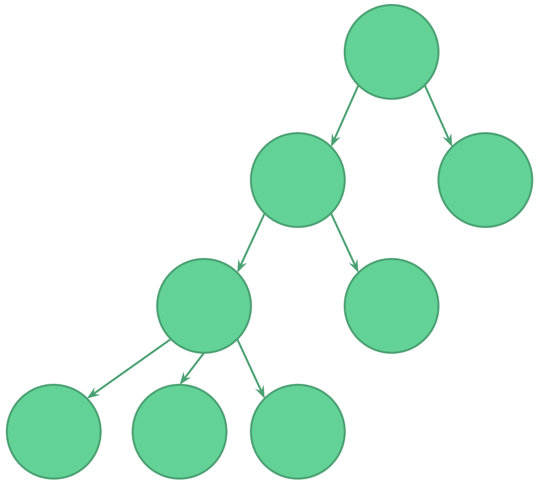
No, all nodes are not connected. It is a forest.

?

Is it a tree?





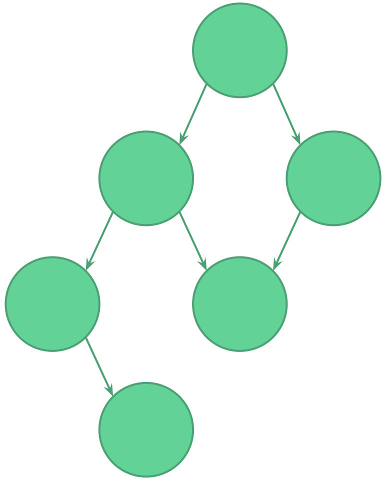


YES

No, all nodes are not connected. It is a forest.

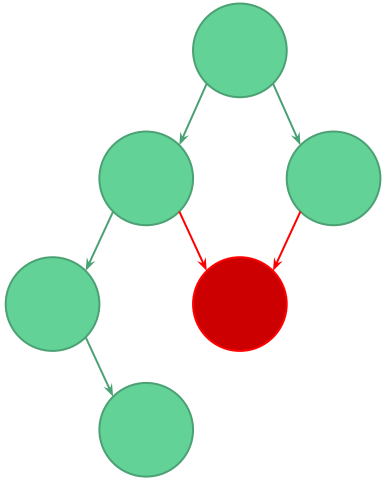
YES

Is it a tree?



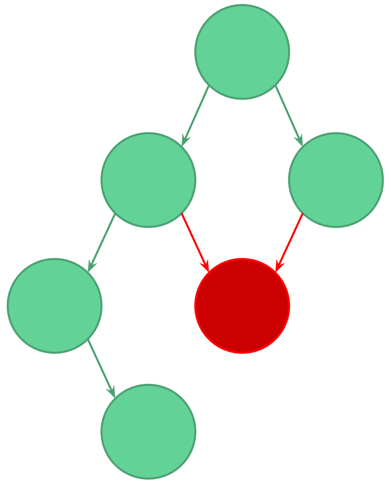
?

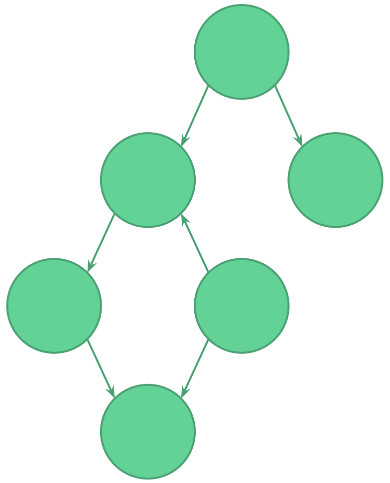
Is it a tree?



No. Each node must have at most one parent

Is it a tree?





No. Each node must have at most one parent

?

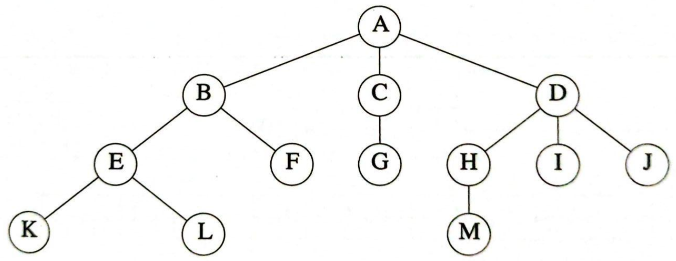
Tree representation

Tree representation

There are several ways to draw a tree besides the one we have seen so far.

- List representation
- Left Child-Right Sibling representation
- Representation as a Degree-Two tree

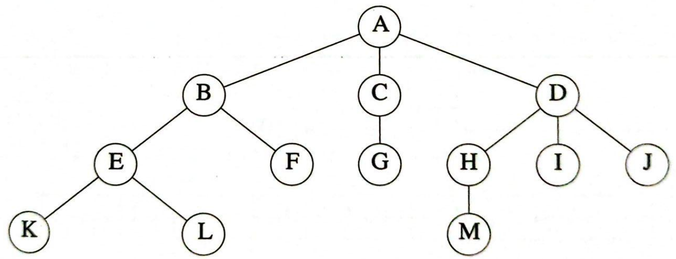
List representation



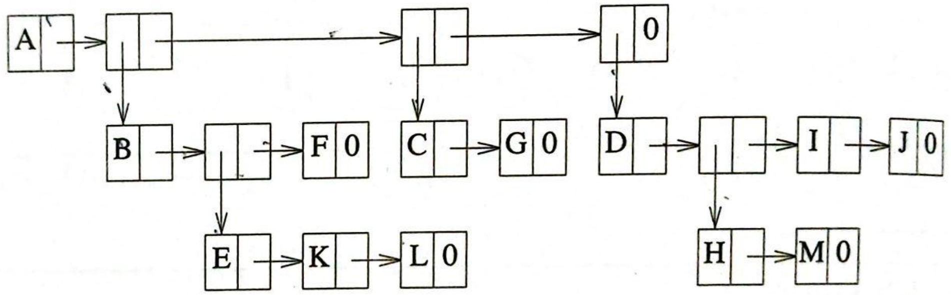
This tree can be written as the list `(A(B(E(K,L),F),C(G),D(H(M),I,J)))`

The information in the root node comes first, followed by a list of the subtrees of that node.

List representation



Memory representation of this tree is as follows



Left child-right sibling representation

One could represent each node by a memory node that has fields for the data and pointers to the tree node’s children.

data	child1	child2	...	child k

Having nodes with varying number of pointers (because the degree of each tree node may be different) makes it difficult to write algorithms.

Having fixed-sized nodes with k pointers, for a tree of degree k, could be one solution but using this node structure is very wasteful of space. Why?

Left Child-Right Sibling representation

This representation requires exactly two pointer per node - one for left child, and one for right sibling.

data	
left child	right sibling

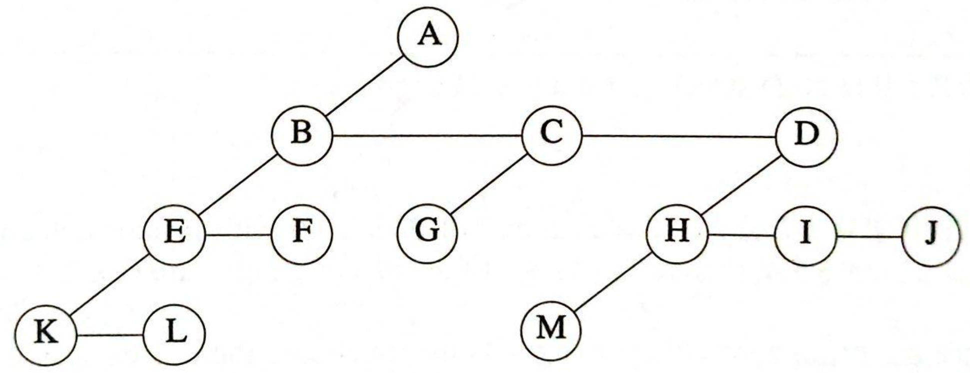


Fig: Left child-right sibling representation of tree of Slide 27

Representation as a Degree-Two tree

Rotate the right-sibling pointers in a left child-right sibling tree clockwise by 45 degrees to obtain the degree-two representation of a tree.

The right sibling (of a left child-right sibling tree) is referred to as the right child.

data	
left child	right child

The right child of the root node of the tree is empty. Why?

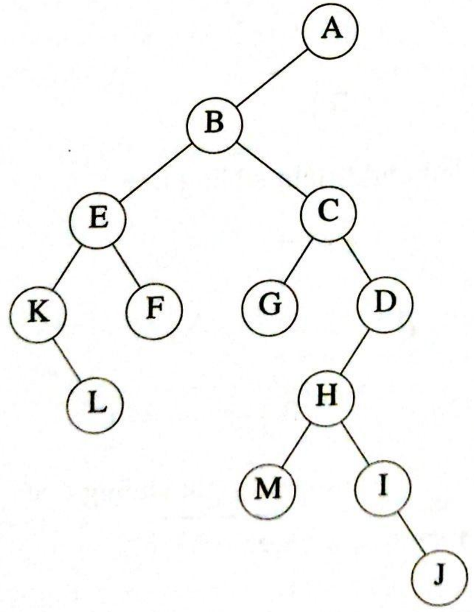


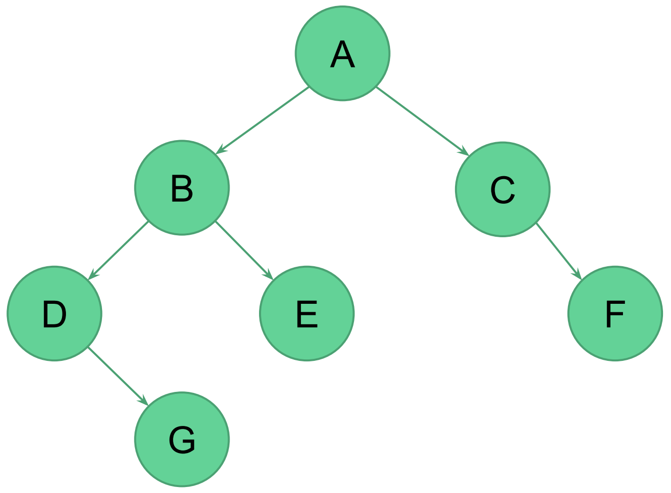
Fig: Degree-two representation of tree of Slide 27

Binary tree

Binary tree

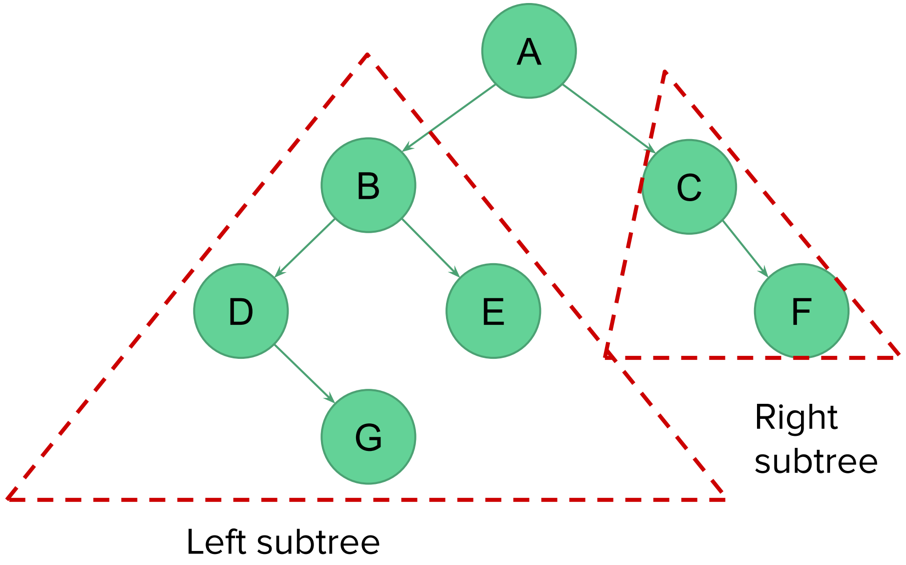
In a binary tree, any node has **at most two children**.

A binary tree may have zero nodes.



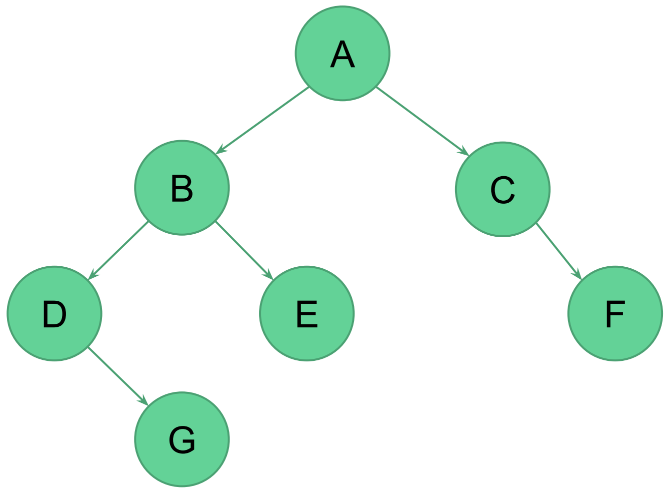
Binary tree

For binary trees, we distinguish between the **left subtree** and the **right subtree.**



Binary tree

A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint (binary) trees called the left subtree and the right subtree.



Properties of a binary tree

1. **Maximum height**

   Given that we need to store N nodes in a binary tree, the maximum height is

   $$H_{max} = N - 1$$

2. **Minimum height**

   Given that we need to store N nodes in a binary tree, the minimum height is

   $$H_{min} = \lfloor log_2 N \rfloor$$

Properties of a binary tree

3. **Minimum number of nodes**

   The minimum number of nodes in a binary tree of height H is

   $$N_{min} = H + 1$$

4. **Maximum number of nodes**

   $$N_{max} = 2^{H+1} - 1$$

Properties of a binary tree

5. **Relation between number of leaf nodes and nodes with two children**

   For any non-empty binary tree, T, if n_0 is the number of leaf nodes and n_2 the number of nodes with two children, then

   $n_0 = n_2 + 1$

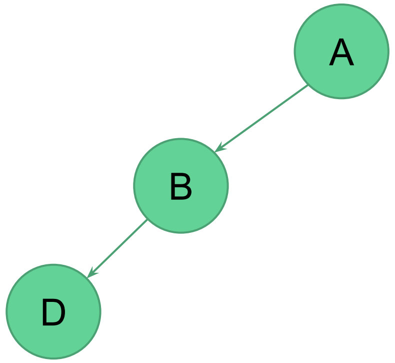
Types of binary tree

1. Skewed binary tree
2. Strictly binary tree
3. Complete binary tree
4. Almost complete binary tree

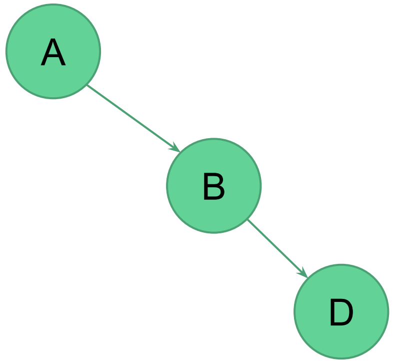
Skewed binary tree

In a skewed binary tree, all nodes have one child or no child at all.

A **left skewed binary tree** has all nodes are having a left child or no child at all.

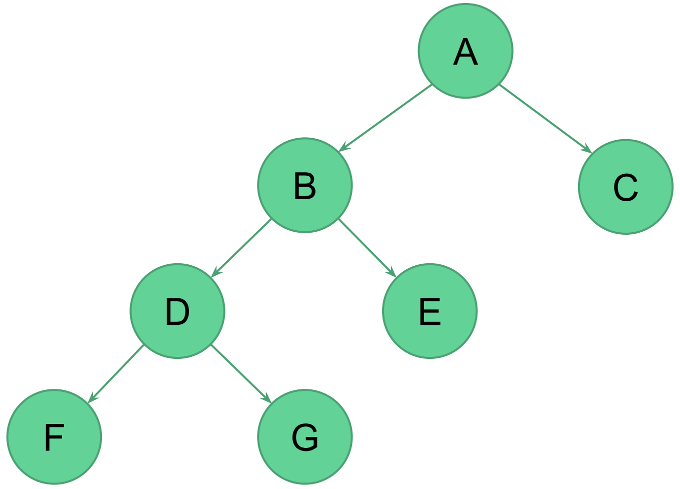


A **right skewed binary tree** has all nodes are having a right child or no child at all.



Strictly binary tree

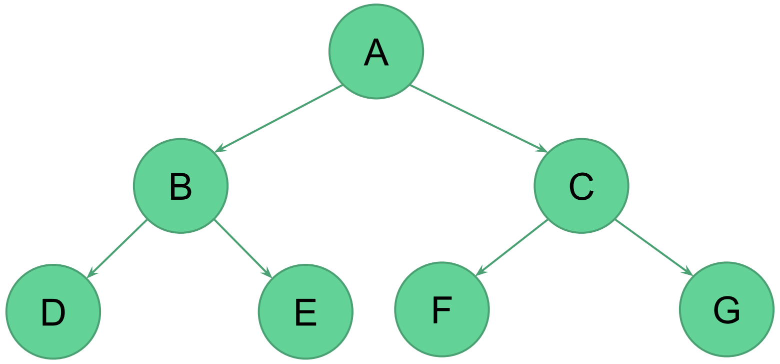
In a strictly binary tree, each node has either 0 or 2 children.



Complete binary tree

A complete binary tree of depth d is the strictly binary tree where all leaves are at level d.

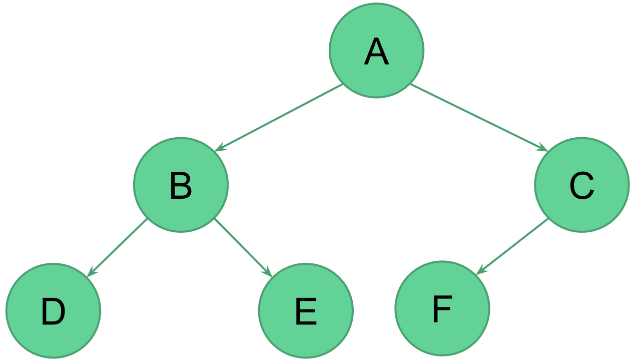
A complete binary tree of height h contains exactly 2^d nodes at depth d, $0 \leq d \leq h$.



Almost complete binary tree

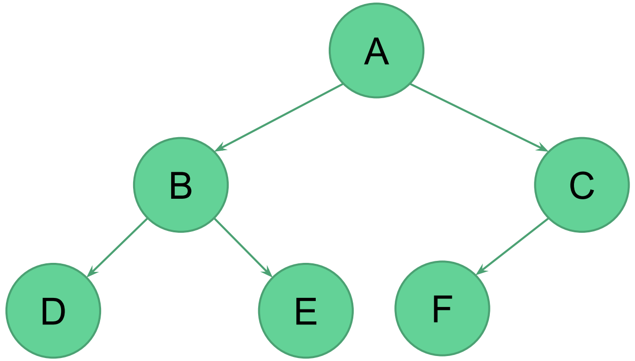
A nearly complete binary tree of height h is a binary tree of height h in which

a) There are 2^d nodes at depth d for d = 1, 2,..., h–1,
b) The nodes at depth h are as far left as possible, i.e.
   If a node p at depth h–1 has a left child, then every node at depth h–1 to the left of p has 2 children. If a node at depth h–1 has a right child, then it also has a left child.



Almost complete binary tree

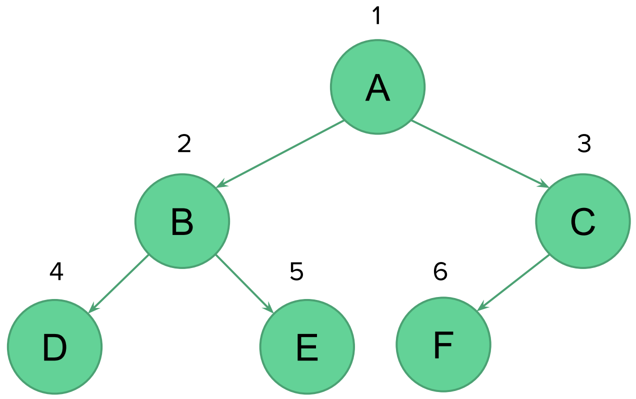
In other words, an almost complete binary tree is a binary tree in which every level of the tree is completely filled except the last level. Also, in the last level, nodes should be attached starting from the left-most position.



Almost complete binary tree

The nodes of an almost complete binary tree can be numbered so that

- the root is assigned the number 1,
- a left child is assigned twice the number assigned to its parent, and
- a right child is assigned one more than twice the number assigned to its parent.



Balanced binary tree

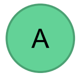
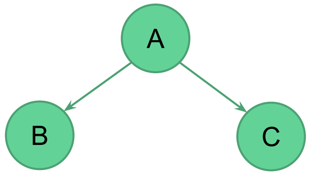
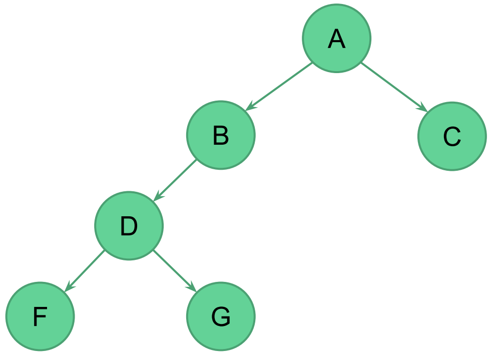
In a balanced binary tree, the height of its subtrees differs by no more than 1, and its subtrees are also balanced.

The difference in height between the left and right subtrees of a binary tree is called its **balance factor**.

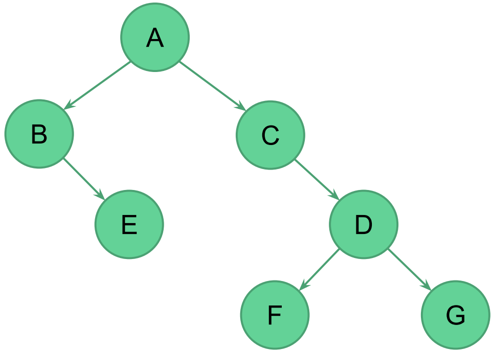
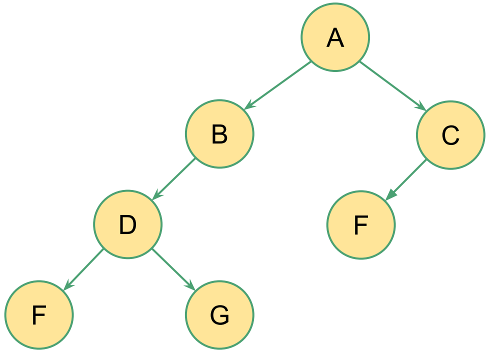
Balance factor, $B = H_L - H_R$

where H_L is the height of the left subtree, and H_R is the height of the right subtree.

Balanced binary tree: Examples

Tree	Balance factor	Balanced?
	0 - 0 = 0	Yes
	1 - 1 = 0	Yes
	3 - 1 = 2	No

Balanced binary tree: Examples

Tree	Balance factor	Balanced?
	?	?
	?	?

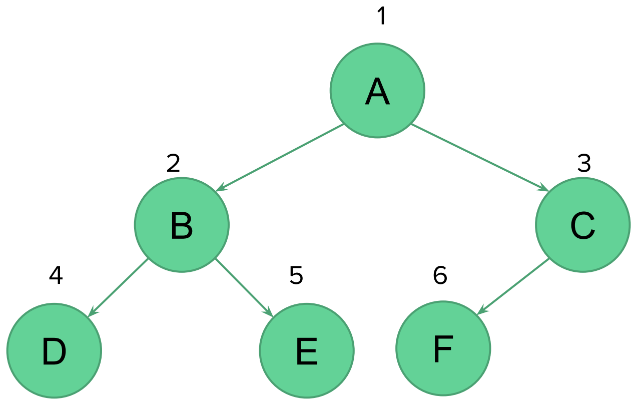
Binary tree representation

1. Array representation
2. Linked representation

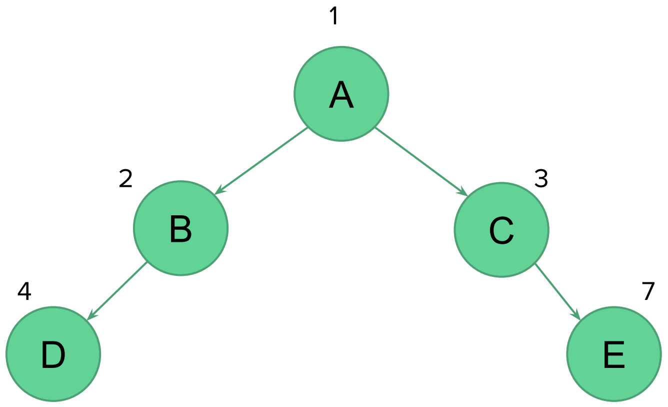
Array representation of a binary tree

- Use one-dimensional array to store the nodes such that position 0 is left empty and the nodes in the binary tree are mapped to the array positions in the following way:
  - If a complete binary tree with n nodes is represented sequentially, then for any node with index i, $1 \leq i \leq n$, we have
    - parent(i) is at $\lfloor i/2 \rfloor$ if $i \neq 1$.
      If $i = 1$, i is at the root and has no parent.
    - Leftchild(i) is at 2i if $2i \leq n$.
      If $2i > n$, then i has no left child
    - Rightchild(i) is at 2i+1 if $(2i+1) \leq n$.
      If $(2i+1) > n$, then i has no right child

Array representation of a binary tree

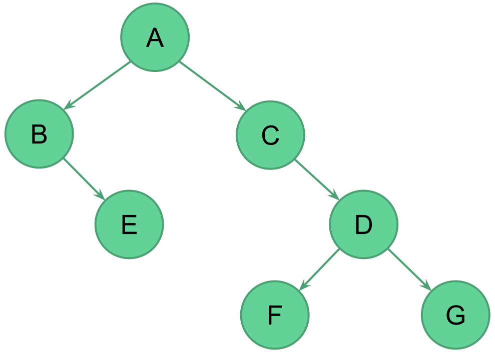


	A	B	C	D	E	F	
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

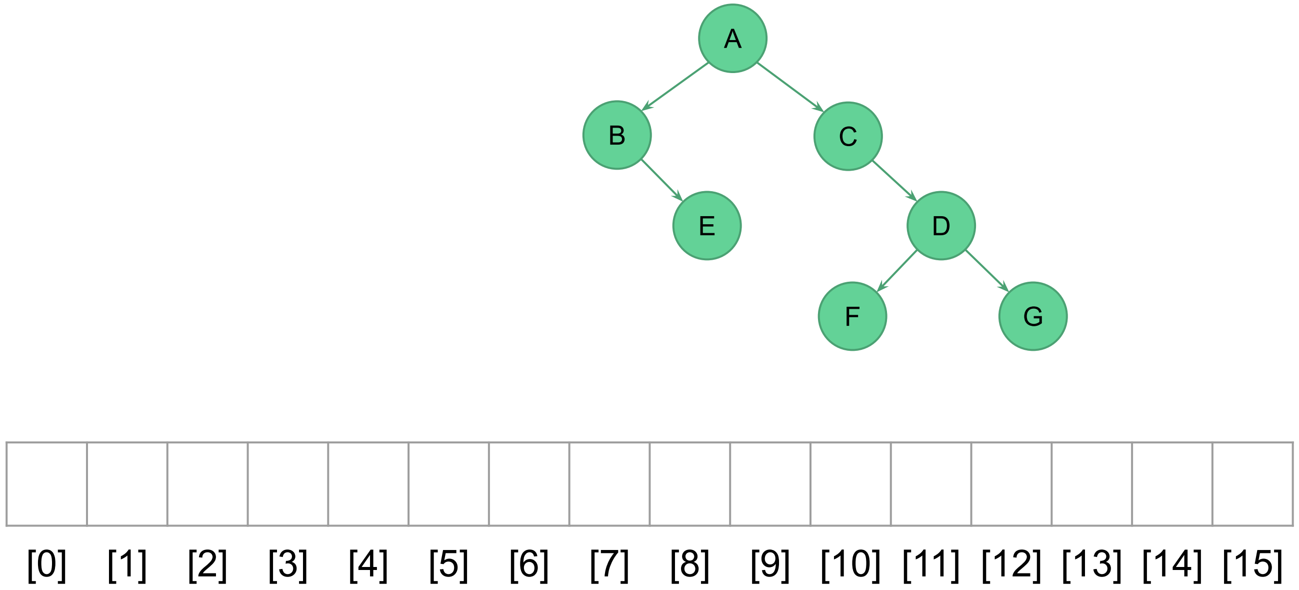


	A	B	C	D			E
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

Array representation of a binary tree



Array representation of a binary tree



Array representation of a binary tree

Advantages

- Direct access to the nodes with the help of the index improves the efficiency
- Useful where the language doesn’t support dynamic memory allocation

Disadvantages

- A lot of unutilized space in most cases.
- Since the array size is limited, enhancement of tree structure is restricted.
- Inefficiency in processing time during insertion or deletion of nodes (from the middle of a tree, which require the movement of potentially many nodes.)

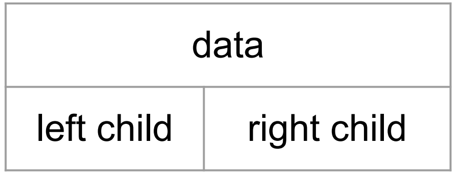
Linked representation of a binary tree

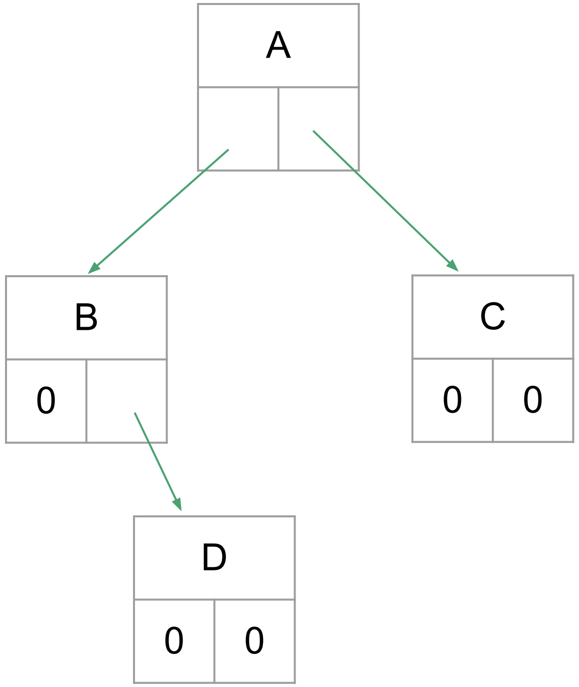
- Although the array representation is good for complete binary trees, it is wasteful for many other binary trees.
- The representation suffers from the general inadequacies of sequential representations.
- Insertion and deletion of nodes from the middle of a tree require the movement of potentially many nodes to reflect the change in level number of these nodes.
- These problems can be overcome easily through the use of a linked representation.

Linked representation of a binary tree

Each node has three fields

1. Left child
2. Data
3. Right child





Linked representation of a binary tree

Advantages

- Efficient memory utilization
- Enhancement of tree is possible
- Insertion and deletion of nodes can be easily handled with the pointers without data movement.

Disadvantages

- It is difficult to implement with the programming language that doesn’t support dynamic memory allocation.
- Since pointers are involved for data reference, more memory might be required.

Tree traversal

Traversing a tree = visiting each node in the tree exactly once

A full traversal produces a linear order for the nodes/information in a tree. This linear order is highly useful in memory management.

Binary tree traversal

Let

L = moving left when at a node

V = visiting the node

R = moving right when at a node

Then there are six possible combinations of traversal:

LVR, LRV, VLR, VRL, RVL, RLV

If we adopt the standard convention that we traverse left before right, then only three traversals remain:

LVR, LRV, and VLR

Binary tree traversal

LVR = Inorder traversal

LRV = Postorder traversal

VLR = Preorder traversal

Inorder traversal (LVR)

Visit the node between its subtrees

- Move down the tree toward the left until you can go no farther
- Then visit the node, move one node to the right and continue
- If you cannot move to the right, go back one more node

Preorder traversal (VLR)

Also known as **Depth-First Traversal (DFT)**

A node is visited before its children

- Visit a node, traverse left and continue
- When you cannot continue, move right and begin again or move back until you can move right and resume

Postorder traversal

Visit the node after its subtrees

Depth-first and breadth-first traversals

Depth-first traversal (DFT)

- All descendants of a child are processed before going on to the next child
- = Preorder traversal

Breadth-first traversal (BFT)

- Each level is completely processed before going to the next level
- The processing proceeds horizontally from the root to all of its children, then to its children's children, and so forth until all nodes have been processed.

Applications of binary trees

- Binary Search Tree (BST)
- Heaps
- Huffman Coding
- Syntax Tree etc.