BCSE202L Data Structures and Algorithms

Non-Linear Data structures- Trees

Overview

Tree Definition, Properties, Terminologies

Binary Tree Traversals

Binary Tree Representation

Binary Search Trees (BST)

Conversion of a non-binary tree to binary tree

Expression Trees

Tree Definition, Properties, Terminologies

Data structure Classification

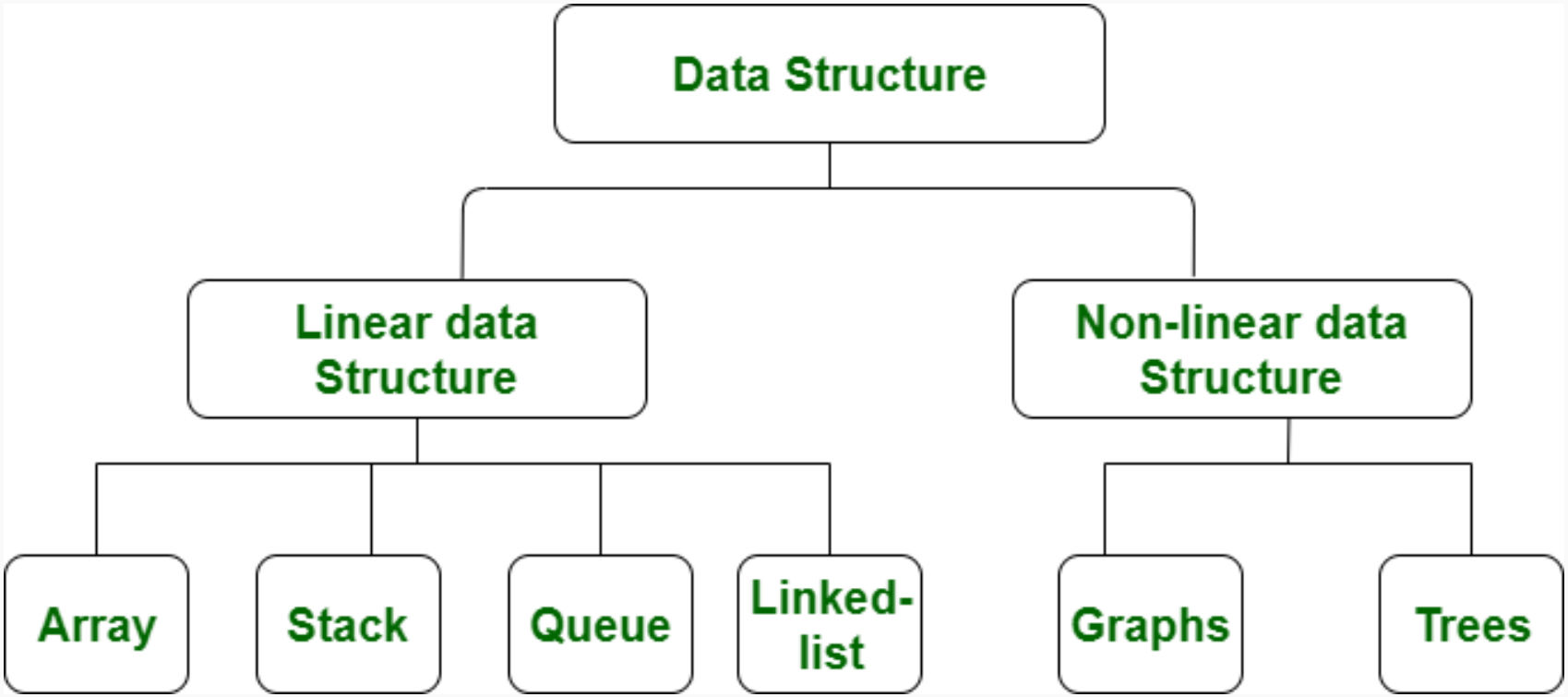


Figure 1: Credit:geeksforgeeks

Graphs

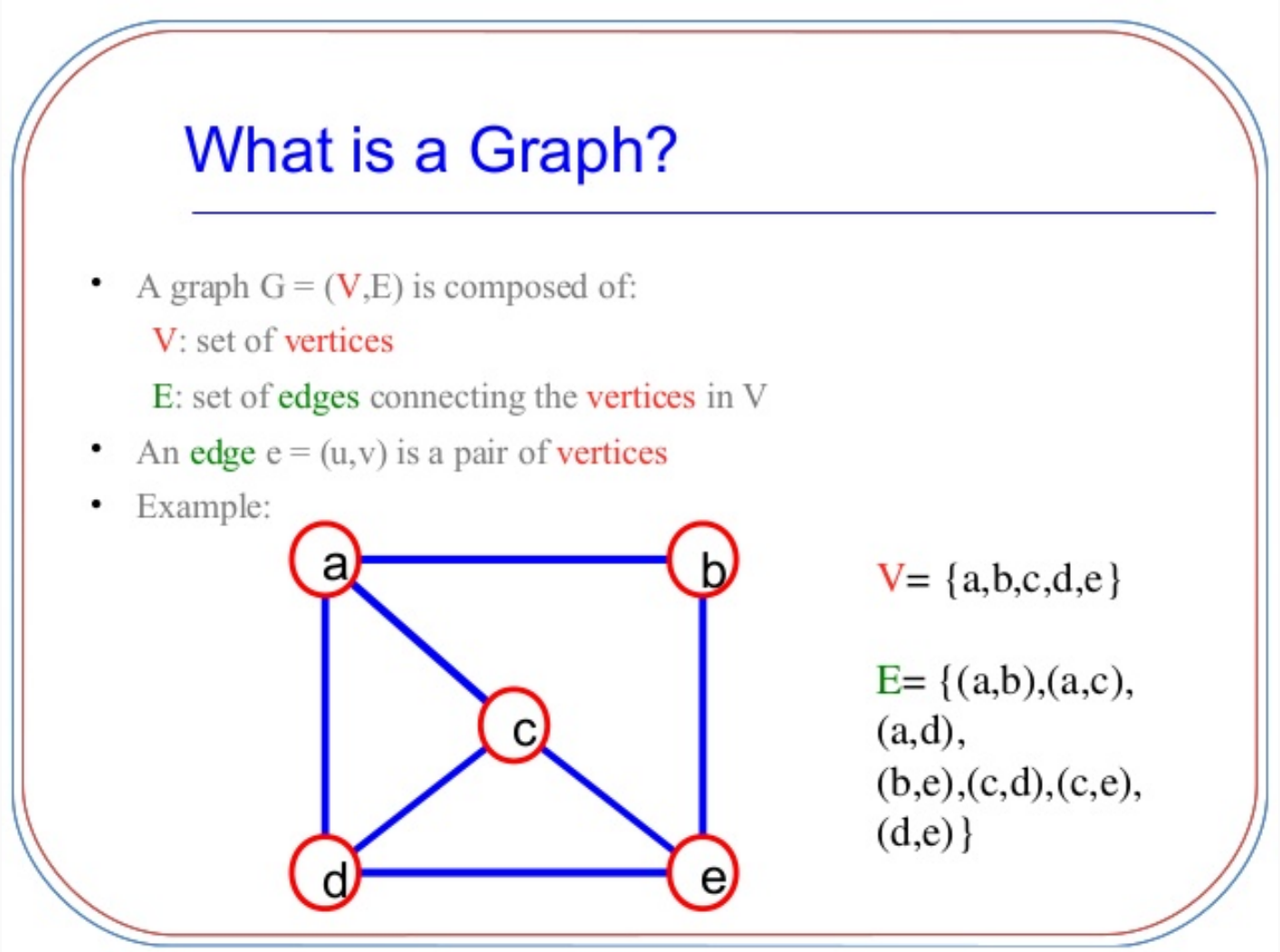


Figure 2: Credit:slideshare.com

Types of Graphs

- Directed and undirected graphs
- Weighted and unweighed graphs

Trees

- Graph Without Cycle

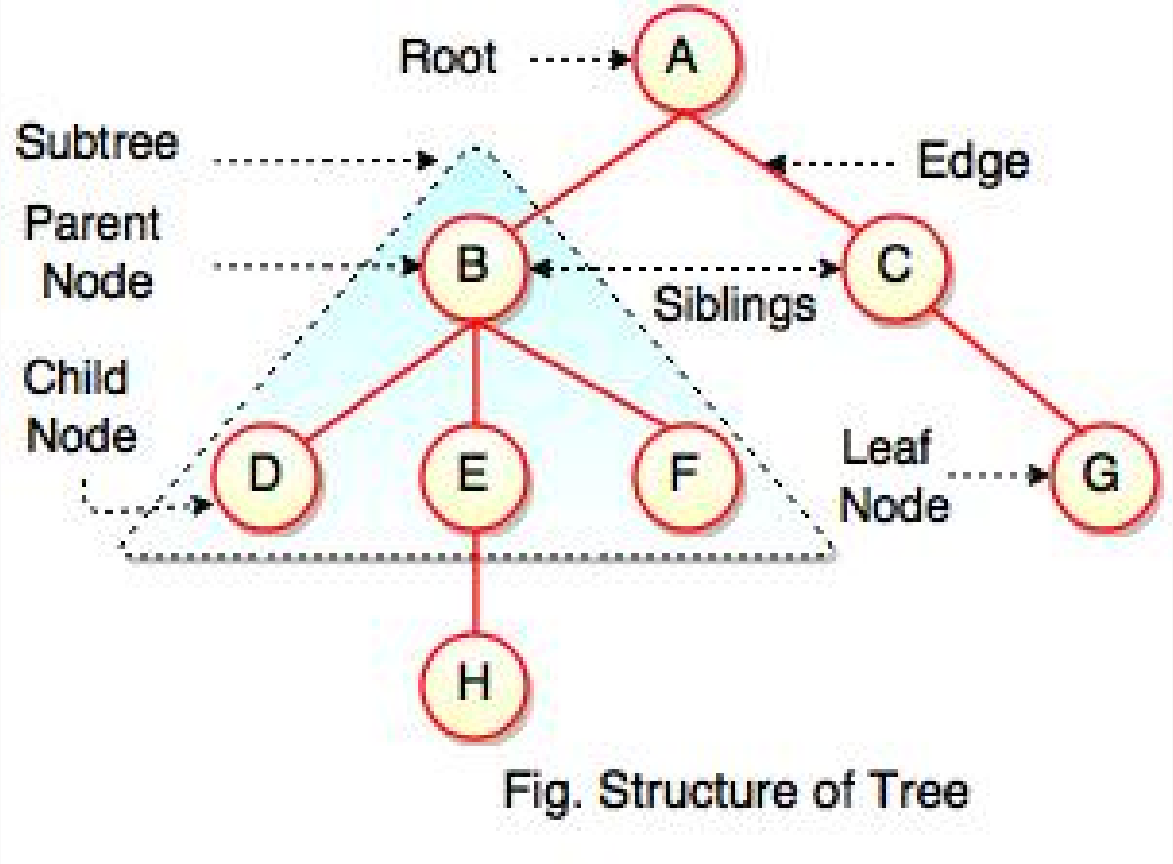


Figure 3: Credit:csestudies.com

Tree Basics

- Root has no parent/ancestor/predecessor
- Leaf has no children/descendant/successor
- Internal nodes– Neither root nor leaf
- Subtree– Descendant of a node with all successors
- Sibling– children of same parent
- Degree- number of children

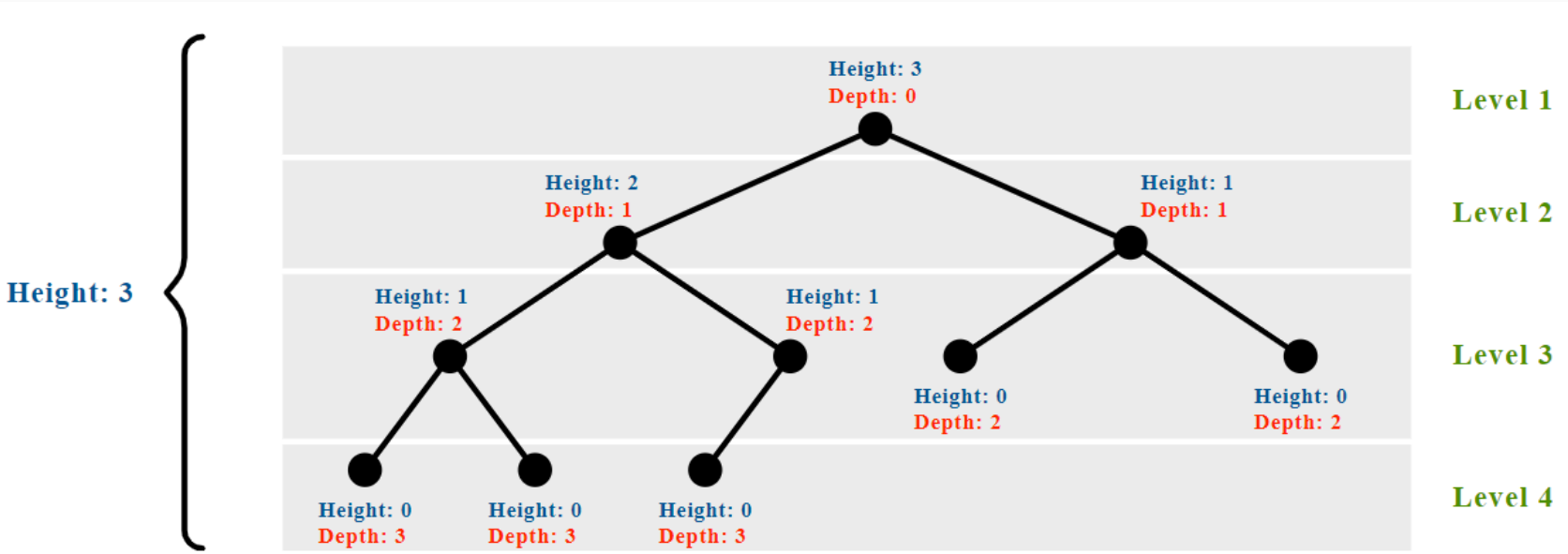
Tree Properties: Height

- Height of a Node
  - Number of edges on the longest path from this node to a leaf
  - Leaves have height 0
- Height of a Tree $\Longrightarrow$ Height of the root

Tree Properties: Depth & Level

- Depth of a Node
  - Number of edges on the path from this node to the root
  - Root has depth 0
- Levels of a Tree
  - A tree has Height + 1 levels
  - Root is at level 1

Height, Depth, & Level



Binary Tree Traversals

Binary Tree

- Every node has max. 2 children (0, 1 or 2)
- Interrelationships
  - Max. No. of nodes possible at any level $i \longrightarrow 2^i$
  - Max. No. of nodes possible in a tree of height $h \longrightarrow 2^{h+1} - 1$
  - Minimum No. of nodes possible in a tree of height $h \longrightarrow h + 1$

Tree Traversals

- Inorder $\longrightarrow$ Left-Root-Right
- Preorder $\longrightarrow$ Root-Left-Right
- Postorder $\longrightarrow$ Left-Right-Root

Inorder Algorithm

1. Traverse the left subtree, i.e., call Inorder(left-subtree)
2. Visit the root
3. Traverse the right subtree, i.e., call Inorder(right-subtree)

- Recurrence: $T(n) = T(n/2) + \Theta(1) + T(n/2)$
- Solving $\longrightarrow T(n) = \Theta(n)$

Preorder Algorithm

1. Visit the root
2. Traverse the left subtree, i.e., call Preorder(left-subtree)
3. Traverse the right subtree, i.e., call Preorder(right-subtree)

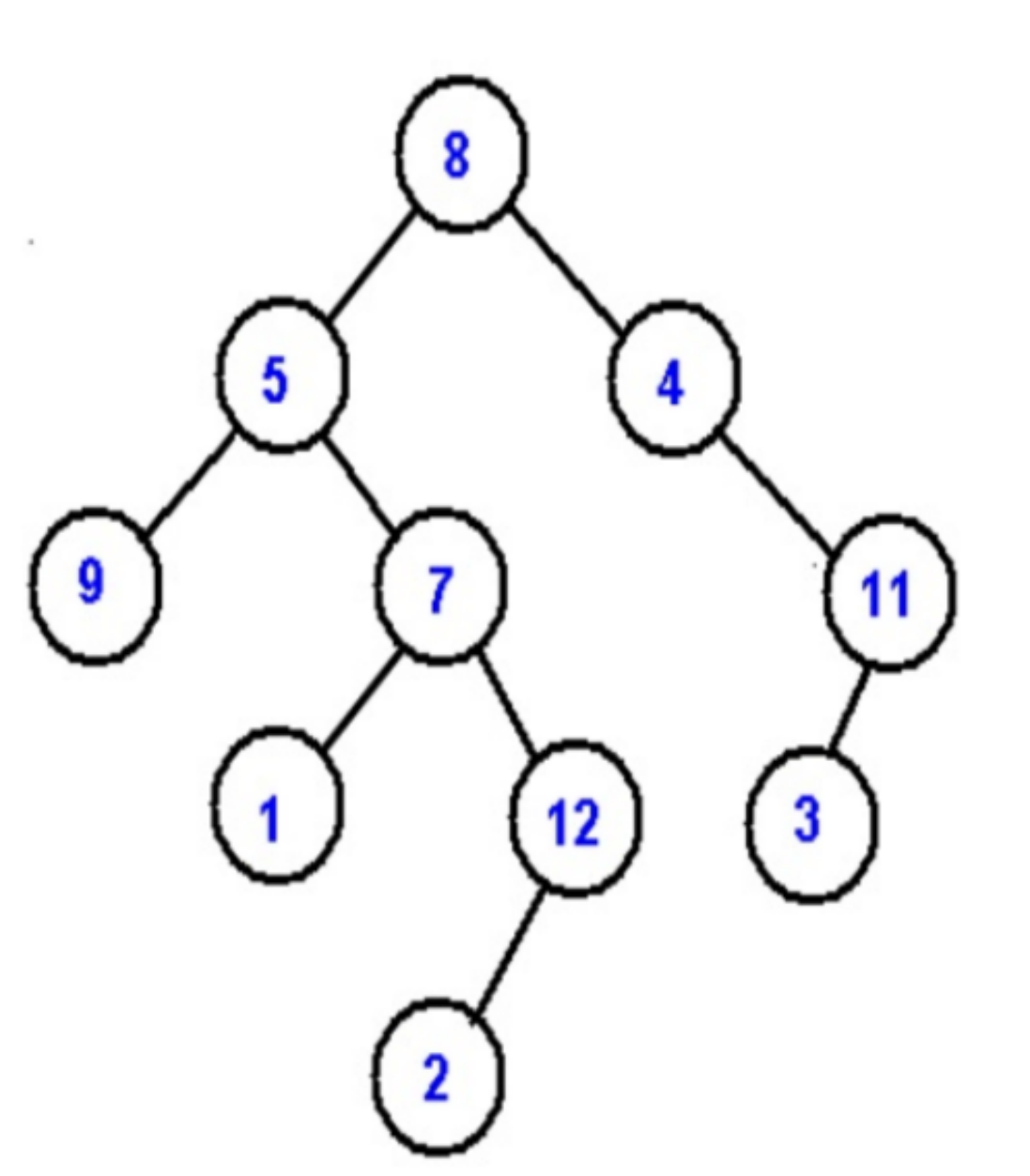
- Recurrence: $T(n) = \Theta(1) + T(n/2) + T(n/2)$
- Solving $\longrightarrow T(n) = \Theta(n)$

Postorder Algorithm

1. Traverse the left subtree, i.e., call Postorder(left-subtree)
2. Traverse the right subtree, i.e., call Postorder(right-subtree)
3. Visit the root

- Recurrence: $T(n) = T(n/2) + T(n/2) + \Theta(1)$
- Solving $\longrightarrow T(n) = \Theta(n)$

Tree Traversals Example



Tree Traversals Answers

PreOrder - 8, 5, 9, 7, 1, 12, 2, 4, 11, 3

InOrder - 9, 5, 1, 7, 2, 12, 8, 4, 3, 11

PostOrder - 9, 1, 2, 12, 7, 5, 3, 11, 4, 8

LevelOrder - 8, 5, 4, 9, 7, 11, 1, 12, 3, 2

Tree from Traversal(s)

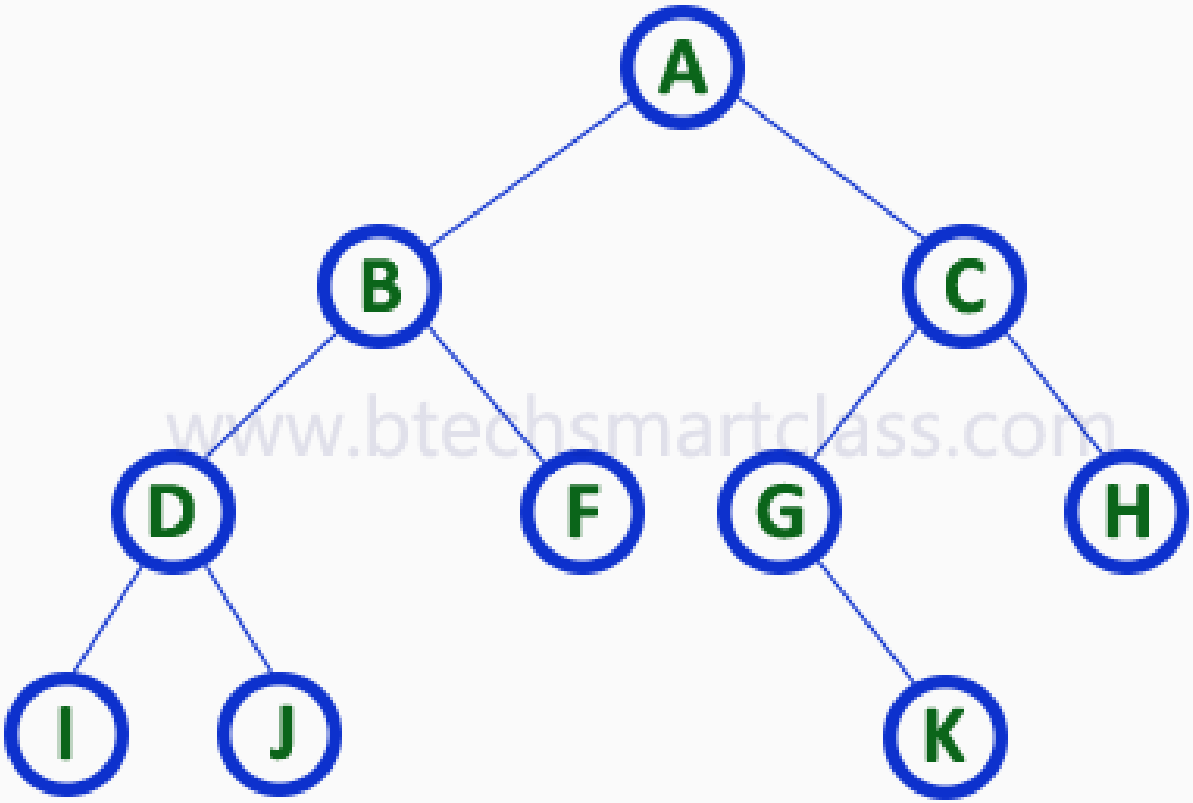
- Can you construct a tree if traversal(s) is/are given?
- If yes, how many traversals (min)?
- Which are they?
- Write an algorithm for this!

Binary Tree Representation

Binary Tree Representation

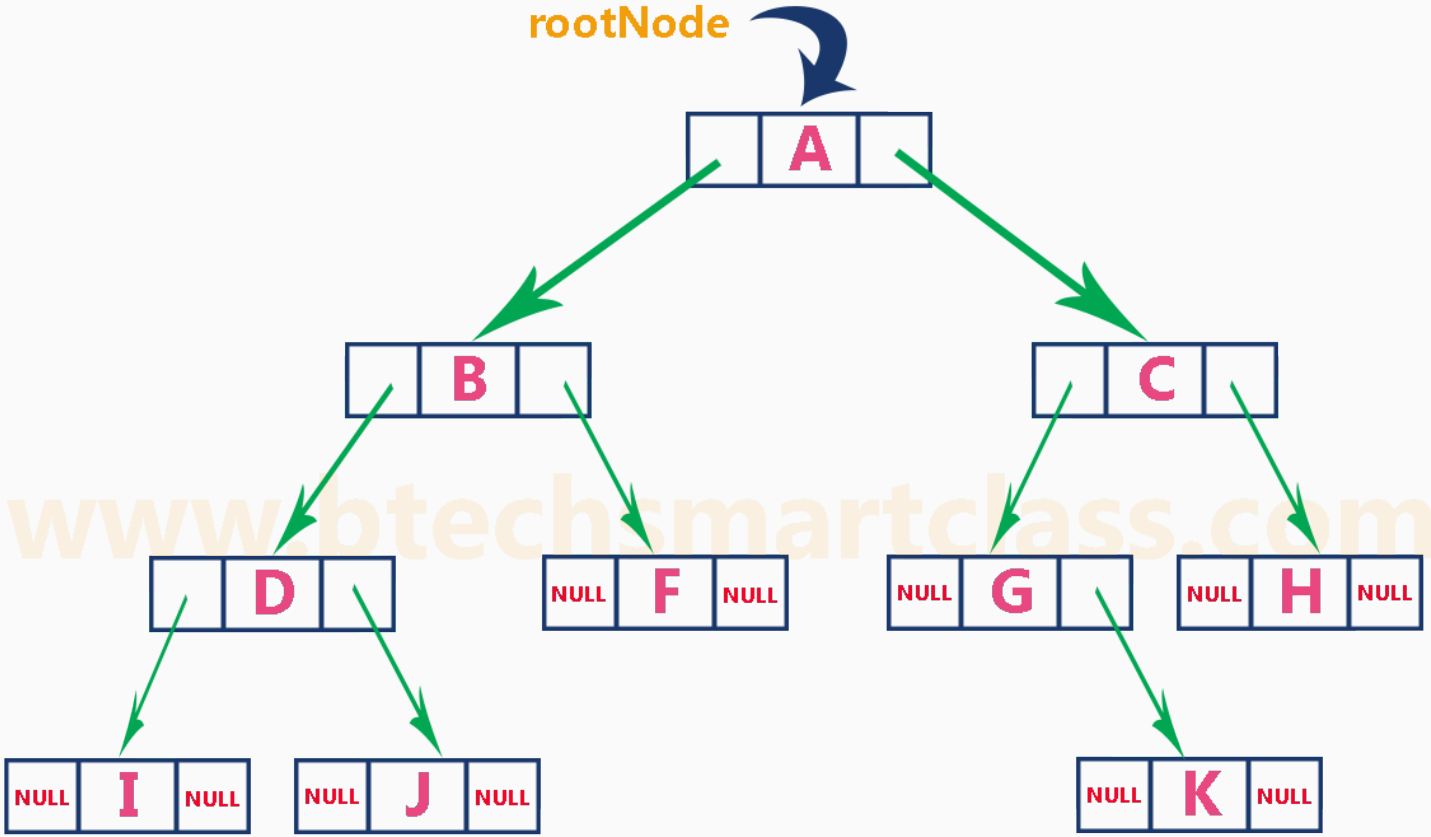
- Array representation
  - 1-D Arrayis used
  - Depth $n \implies$, we need an array of max size, $2n + 1$
- Linked list representation

Array Representation



A	B	C	D	F	G	H	I	J	-	-	-	K	-	-	-	-	-	-	-	-

Linked list Representation

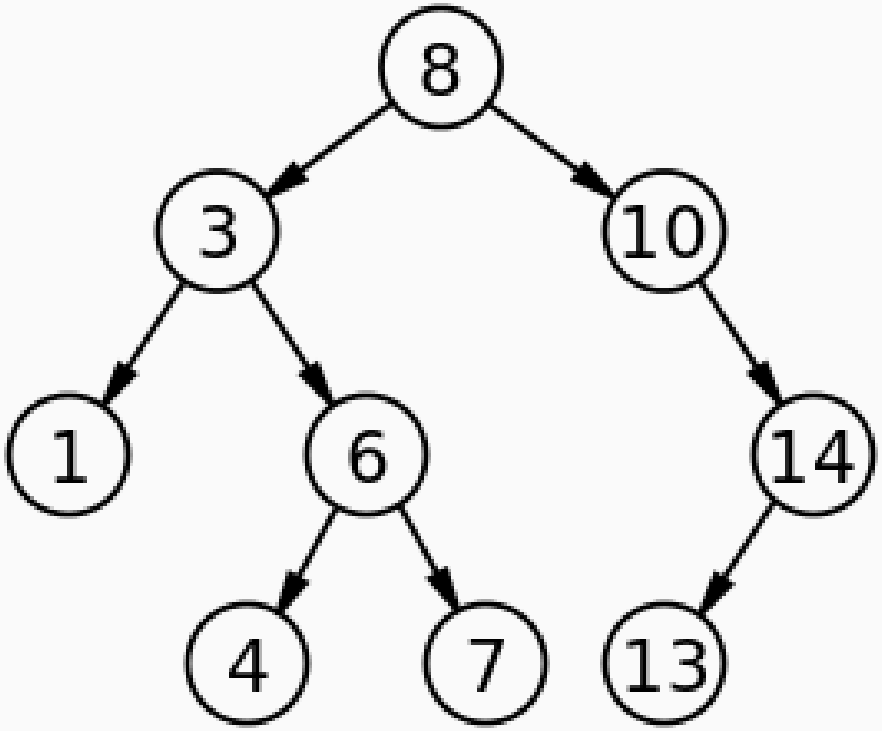


Binary Search Trees (BST)

Binary Search Trees (BST)

- Binary Search Tree is a binary tree with the following properties:
  1. The left subtree of a node contains only nodes with keys lesser than the node's key.
  2. The right subtree of a node contains only nodes with keys greater than the node's key.
  3. The left and right subtree each must also be a binary search tree.
  4. Obviously keys must be distinct/duplicates not allowed

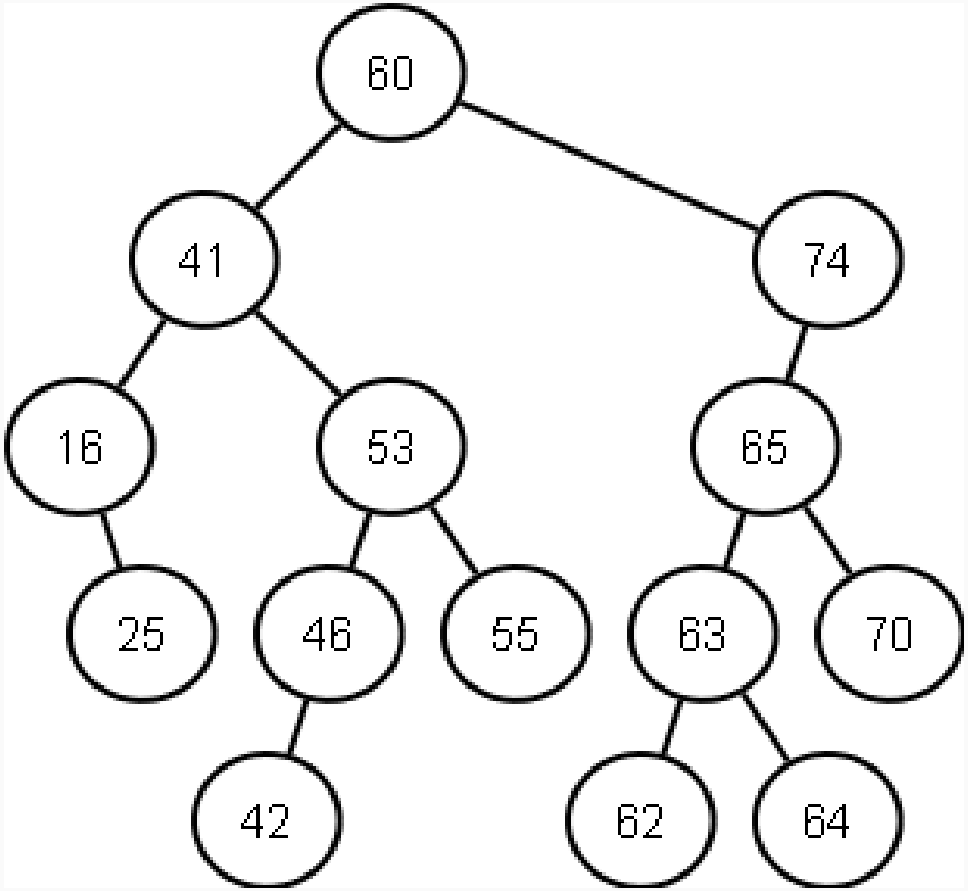
BST Example



BST Creation/Insertion

- Construct BST with the sequence:
- 60, 41, 74, 16, 53, 65, 25 46, 55, 63, 70, 42, 62, 64

BST Answer



What is the inorder traversal of a BST? Check it out

BST Deletion

- Three cases
  1. No Child:- Simply delete – Eg. Delete 25 or 64
  2. One Child:- Replace it with its only child– Eg. Delete 74 or 16
  3. Two Children:- 2 Sub-cases (any one– not both):
     - 3.1 Replace with inorder predecessor $\Longrightarrow$ Largest of the left subtree – Eg.:- delete 60
     - 3.2 Replace with inorder successor $\Longrightarrow$ Smallest of the right subtree – Eg. delete 41
- Analysis: insertion/search/deletion: $O(h)$, where h is the height

Balanced and Skewed Trees

- In the previous example we got almost a height balanced tree
- Construct a BST for 1, 2, 3, 4, 5
- This will be a right-skewed tree
- Construct a BST for 5, 4, 3, 2, 1
- This will be a left-skewed tree
- Now construct for 3, 2, 4, 1, 5
- Will be a height-balanced one
- So for the same data we get different BSTs depending on the order of elements

BSTs

- What is the number of BSTs possible with n nodes?
- For unlabelled nodes: $\frac{\binom{2n}{n}}{(n+1)}$
- For labelled nodes: $\frac{\binom{2n}{n} * n!}{(n+1)}$
- Why/How?
- Explore and find it out!!

Exercise

1. Draw a binary search tree (BST) of the elements accepted in the following order: 14, 4, 5, 13, 15, 11, 6, 2, 9, 3, 8, 10, 12, 7, 19. Perform deletions from the BST in the following order: 7, 13, 14, 4.

Finding the k^{th} smallest element in a BST

- The Inorder Traversal of a BST traverses the nodes in increasing order.
- So the idea is to traverse the tree in Inorder.
- While traversing, keep track of the count of the nodes visited.
- If the count becomes k, print the node.
- Exercise:-Write a pseudocode for this

Conversion of a non-binary tree to binary tree

Conversion of a non-binary tree to binary tree

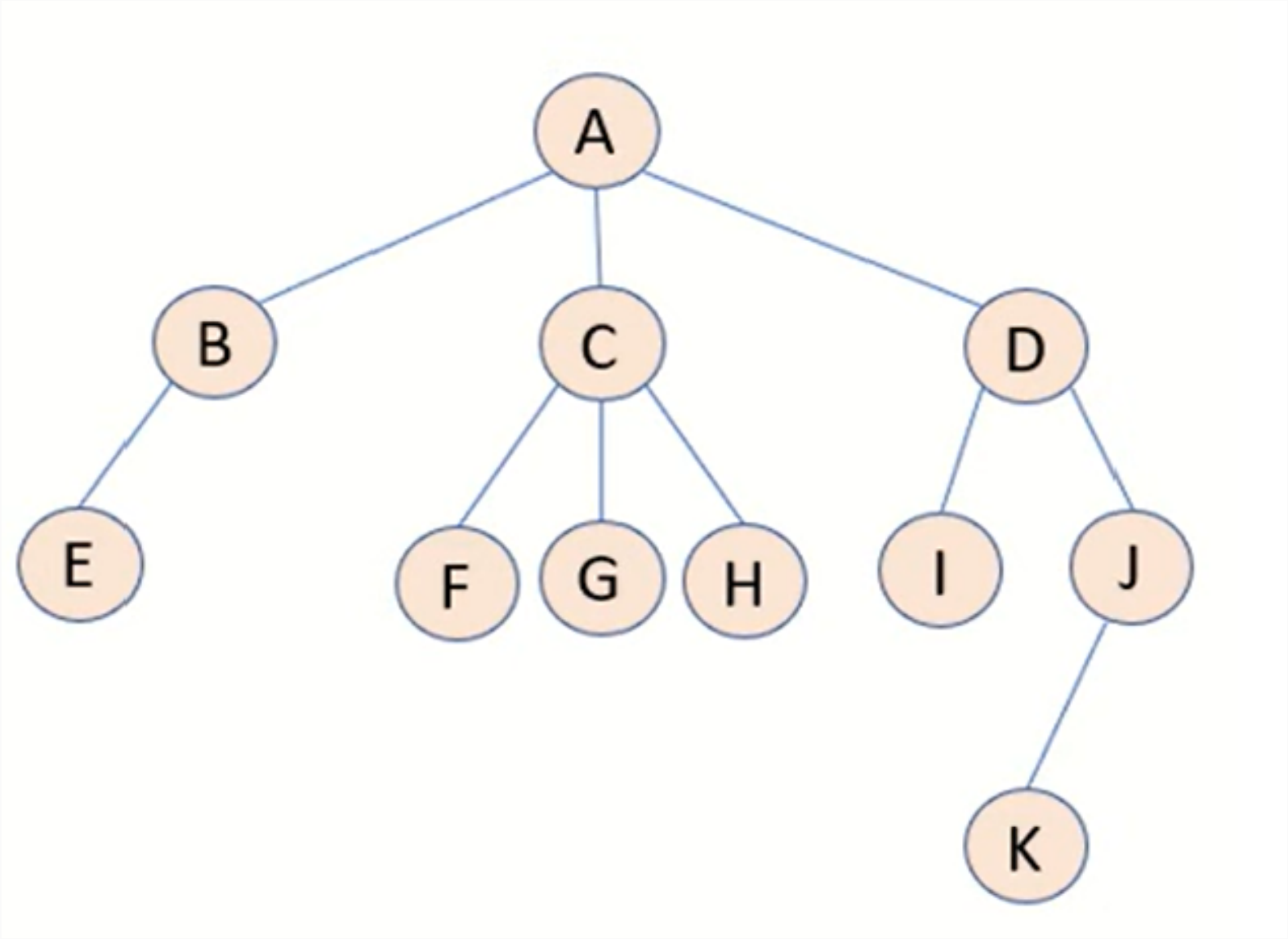
- Any general tree would be n-ary tree
- Because any node can have max. n children
- Binary tree is a special case of n-ary tree

Conversion Algorithm

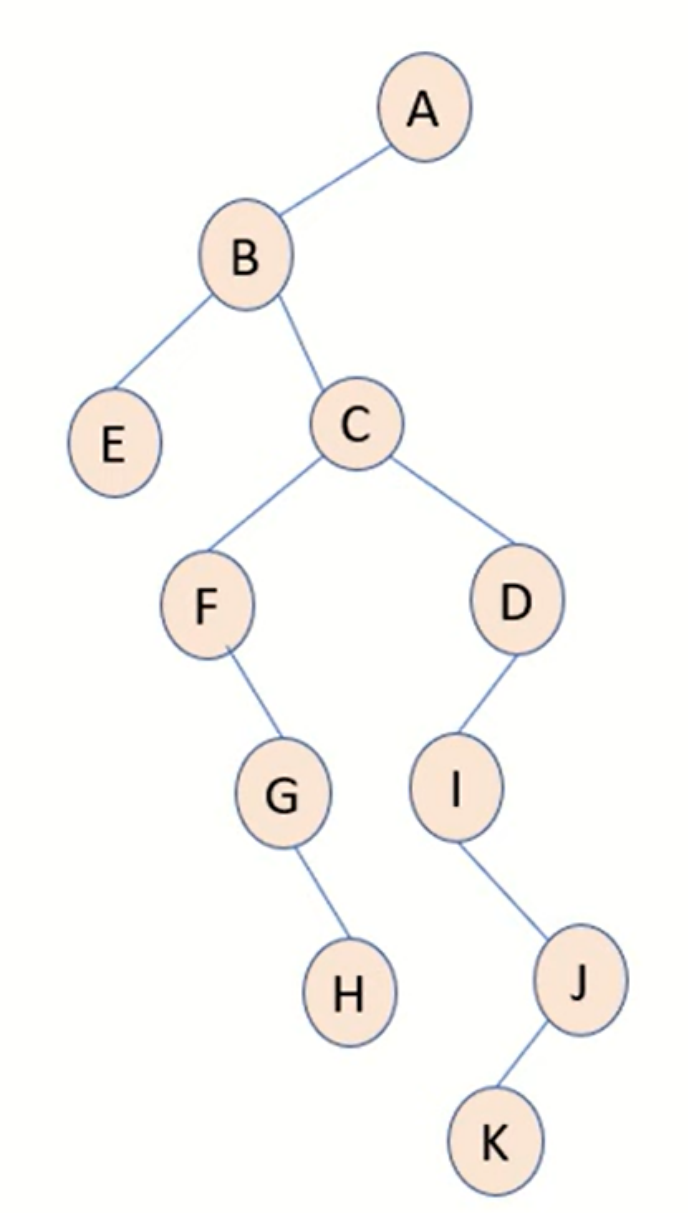
1. Root of binary tree = Root of general tree
2. Left child of any node in binary tree = Leftmost child of the node in the general tree
3. Right child of any node in binary tree = Right sibling of the node in the general tree

Example

- Convert to binary tree



Answer



Expression Trees

Expression Trees

- Recall: Expressions: Operators and operands arranged based on specific rules
- Infix, prefix and postfix
- Representation of an expression in a binary tree $\longrightarrow$ Expression tree
- Leaves/External nodes $\longrightarrow$ Operands
- Non-leaf nodes/Internal nodes $\longrightarrow$ Operators

Expression Tree Demo

- Recall precedence and associativity of operators
- Demo:- $a * b/c + e/f * g + k - x * y$
- Operands at leaf
- More precedence operator to be evaluated first; so, should come near/next to leaves
- Lesser precedence operator to be evaluated towards last; so, should come towards the root

Verification

- Checking correctness:
- Inorder of infix expression tree $\longrightarrow$ Infix
- Preorder of infix expression tree $\longrightarrow$ Prefix
- Postorder of infix expression tree $\longrightarrow$ Postfix
- Exercise: Expression tree with parentheses:-$(5 - x) * y + 6/(x + z)$

Summary

Tree Definition, Properties, Terminologies

Binary Tree Traversals

Binary Tree Representation

Binary Search Trees (BST)

Conversion of a non-binary tree to binary tree

Expression Trees

Thank you..!!