Made By:
Kenan Gazwan

Trees

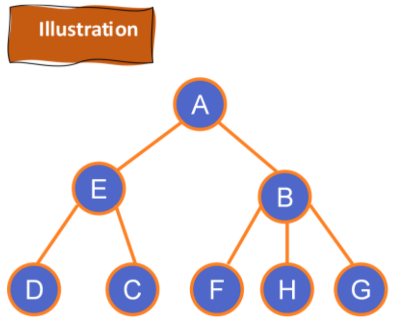
ICS202-Summary

King Fahd University of Petroleum and Minerals

(Trees)

✓ Definition of a Tree

A tree is a finite set of nodes together with a finite set of edges (arcs) that define parent-child relationships.



Nodes = {A, B, C, D, E, F, G, H}

Edges = {(A, E), (A, B), (E, D), (E, C), (B, F), (B, H), (B, G)}

- An edge is a connection between a parent and its child.
- A path from one node to another is a list of nodes such that each is the parent of the next node in the list.
- The length of the path = number of edges

Trees Properties:

- ✓ It has one designated node, called the root, that has no parent.
- ✓ Every node, except the root, has exactly one parent.
- ✓ A node may have zero or more children.
- ✓ There is a unique directed path from the root to each node.

✓ Tree Terminology

Ordered tree: A tree in which the children of each node are linearly ordered (usually from left to right).

Ancestor of a node *v*: Any node, including *v* itself, on the path from the root to the node.

Proper ancestor of a node *v*: Any node, excluding *v*, on the path from the root to the node.

Descendant of a node v: Any node, including v itself, on any path from the node to a leaf node.

Proper descendant of a node v: Any node, excluding v, on any path from the node to a leaf node.

Subtree of a node v: A tree rooted at a child of v.

Degree: the number of subtrees of a node (the number of children).

Leaf: a node with degree 0 (has no children)**.**

Nonterminal or internal node: a node with degree greater than 0**.**

Siblings: nodes that have the same parent**.**

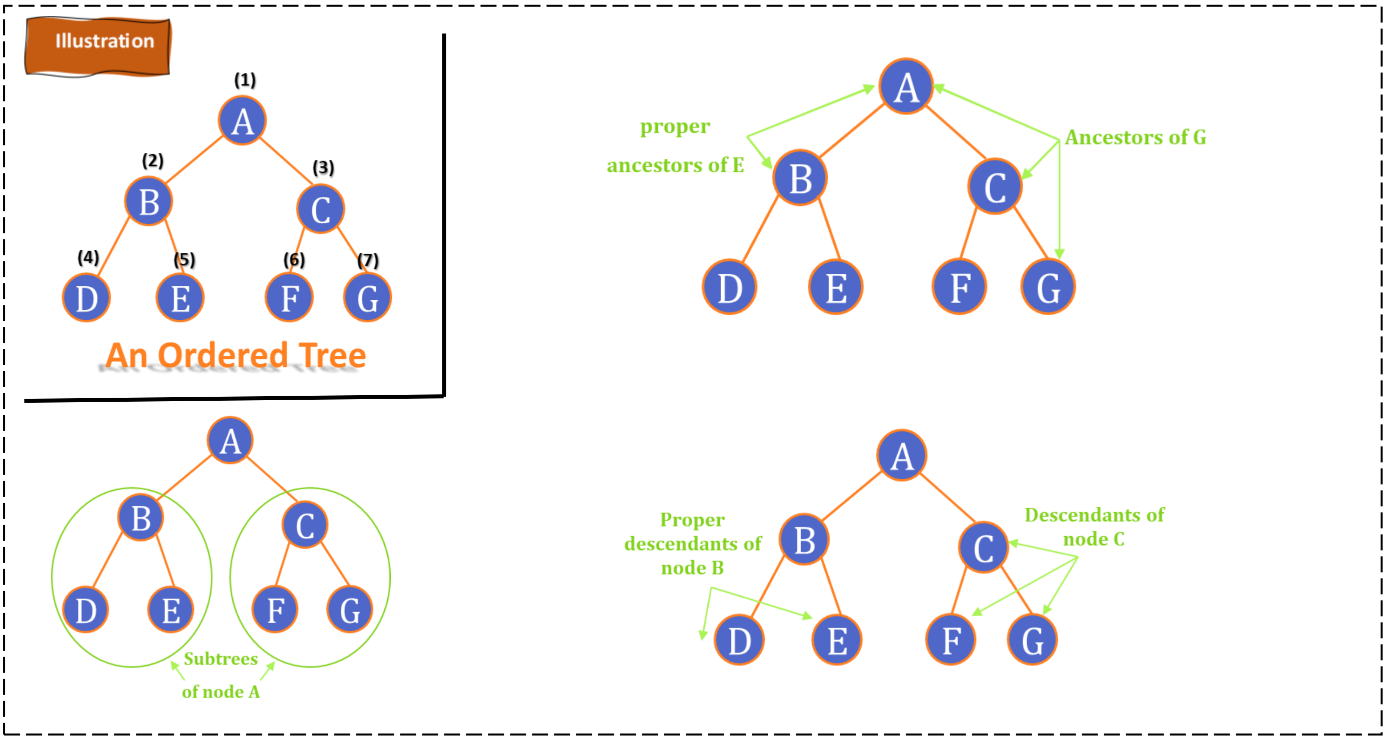
Size: the number of nodes in a tree**.**

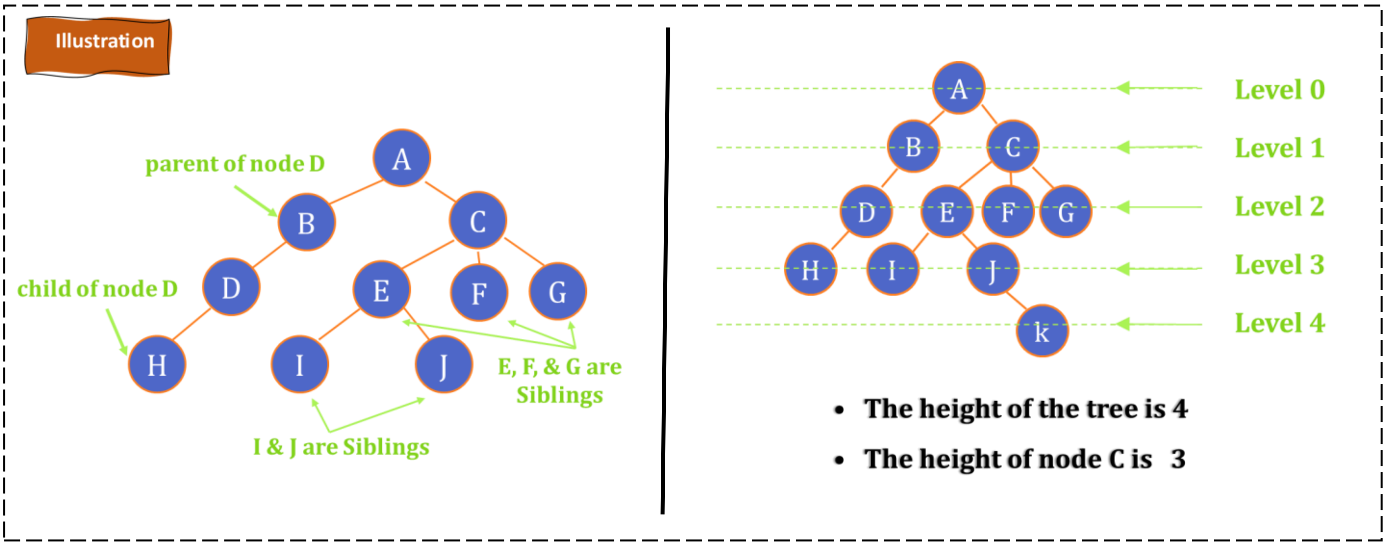
Level (or depth) of a node v: The length of the path from the root to node v.

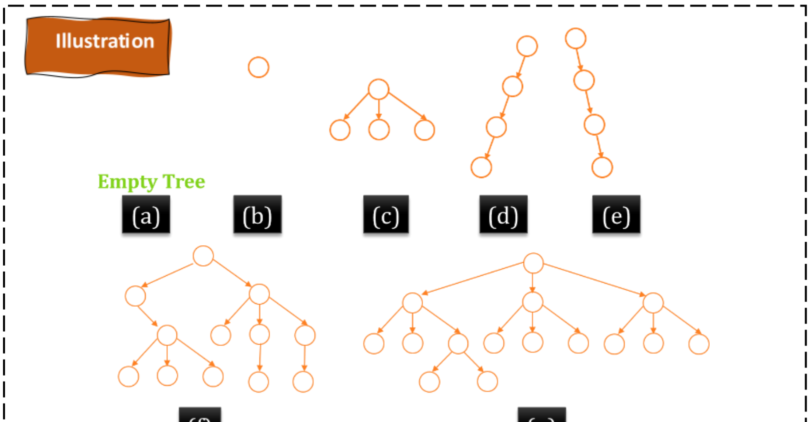
[Remember!! | length of path = number of edges**]**

Height of a node v: The length of the <u>longest</u> path from v to a leaf node.

- The height of a nonempty tree is the height of its root.
- The height of an empty tree is -1.







Examples of Trees

✓ Importance of Trees

Trees are very important data structures in computing. They are suitable for:

- **Hierarchical structure representation**
  - File directory
  - Organizational structure of an institution
  - Class inheritance tree
- **Problem representation**
  - Expression tree
  - Decision tree
- **Efficient algorithmic solutions**
  - Search trees
  - Efficient priority queues via heaps

✓ General Trees & Their Representations

There is no limit to the number of children that a node can have.

General Tree as Linked List:

Representation 1:

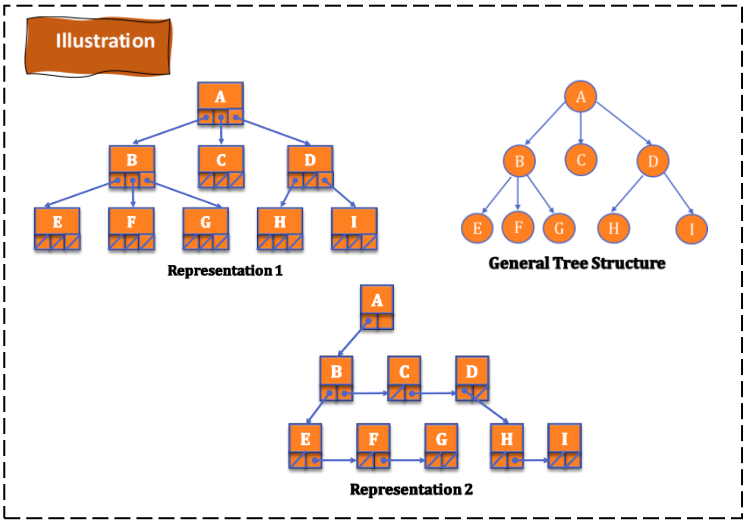
Each Node Consists of:

- Ref to Subtree 1 (Child)
- Ref to Subtree 2 (Child)
- Ref to Subtree 3 (Child)
- Data

Representation 2:

Each Node Consists of:

- Ref to Subtree (Child)
- Ref to Next (Sibling)
- Data



✓ N-ary Trees

It follows that the degree of each node in an N-ary tree is at most N.

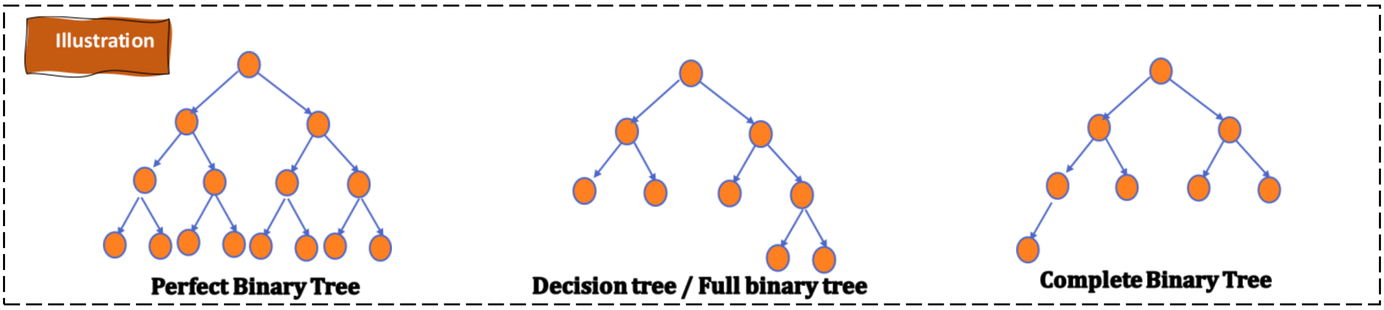
- ✓ 2-ary (binary) tree
- ✓ 3-ary (ternary) tree

✓ Binary Tree

- A binary tree is an N-ary tree for which N = 2
- A binary tree is either:
  1. An empty tree, or
  2. A tree consists of a root node and at most two non-empty binary subtrees.

Classification of Binary Tree:

- **Perfect binary tree**
  - A binary tree in which each level k, $k \geq 0$, has 2^k nodes.
  - All the leaf nodes are at the same depth, and all non-leaf nodes have two children.
- **Decision tree (full binary tree)**
  - A binary tree in which every node is either a leaf node or an internal node with two children.
  - *Num of leaves = Num of internal nodes + 1*
- **Complete binary tree**
  1. Each level k, $k \geq 0$, other than the last level, contains the maximum number of nodes for that level, that is 2^k.
  2. The last level <u>may</u> or <u>may not</u> contain the maximum number of nodes.
  3. The nodes in the last level are filled from left to right.



Binary Tree Properties of *n*-nodes and height-*h*:

Maximum Height	$n - 1$
Minimum Height	$[log\ (n+1)] - 1$
Maximum Number of Nodes	$2^{h+1} - 1$
Minimum Number of Nodes	$h + 1$

Binary Tree Traversal:

The process of systematically <u>visiting</u> all the nodes in a tree and <u>performing</u> a certain process.

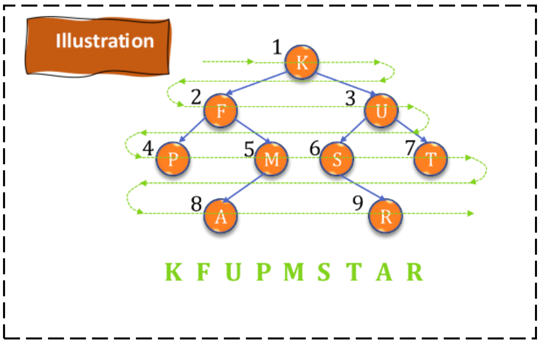
A traversal starts at the root of the tree and visits every node in the tree exactly <u>once</u>.

Traversal Methods:

1. Breadth-First Traversal (or Level-order Traversal).
2. Depth-First Traversal:
    - Pre-order traversal
    - In-order traversal (for binary trees only)
    - Post-order traversal

1) Breadth-First Traversal

- Breadth-First Process:
    - Start from the top node.
    - Then go one level down.
    - Go through all of the children nodes from left to right
    - Repeat the process until every level is visited.



- Breadth-First Implemented by Queue:
    - Enqueue the root node of the tree.
    - While the queue is not empty:
        - Dequeue a node from the queue
        - Output the data field of this node.
        - If the left child of the node is not empty: Enqueue the left child.
        - If the right child of this node is not empty: Enqueue the right child.

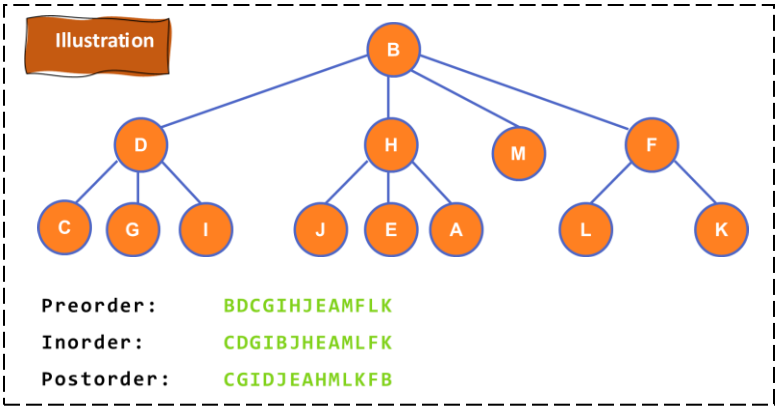
2) Depth-First Traversal

Name	*The Process for Each Node*	*Code*

Preorder (N-L-R)	Visit the node.
	Visit the left subtree, if any
	Visit the right subtree, if any
		`void preorderTraversal(Node node) {`
		`if(node == null)`
		`return;`
		`System.out.print(node.data + " ");`
		`preorderTraversal(node.left);`
		`preorderTraversal(node.right);`
		`}`

Inorder (L-N-R)	Visit the left subtree, if any .
	Visit the node.
	Visit the right subtree, if any.
		`void inorderTraversal(Node node) {`
		`if(node == null)`
		`return;`
		`inorderTraversal(node.left);`
		`System.out.print(node.data + " ");`
		`inorderTraversal(node.right);`
		`}`

Postorder (L-R-N)	Visit the left subtree, if any
	Visit the right subtree, if any
	Visit the node.
		`void postorderTraversal(Node node) {`
		`if(node == null)`
		`return;`
		`postorderTraversal(node.left);`
		`postorderTraversal(node.right);`
		`System.out.print(node.data + " ");`
		`}`

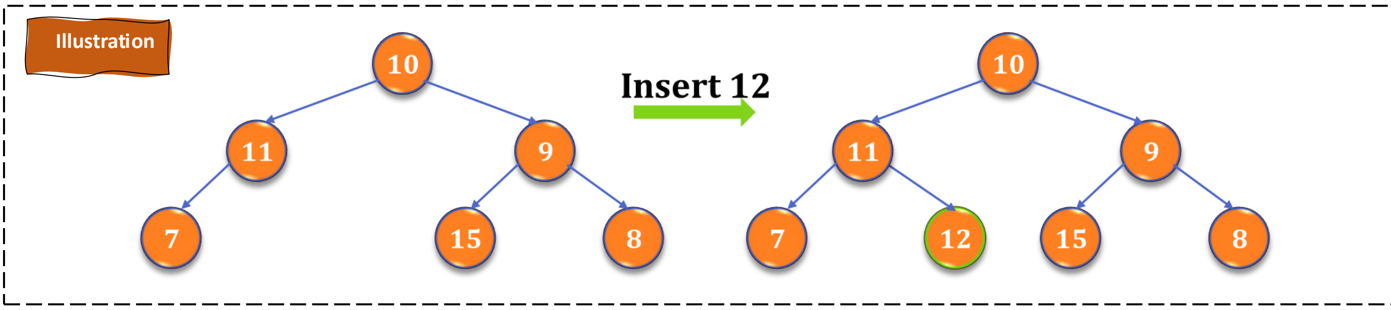


Traversing Expression Trees:

- A <u>Preorder</u> traversal produces a <u>prefix</u> expression.
- An <u>Inorder</u> traversal produces an <u>infix</u> expression.
- A <u>Postorder</u> traversal produces a <u>postfix</u> expression.

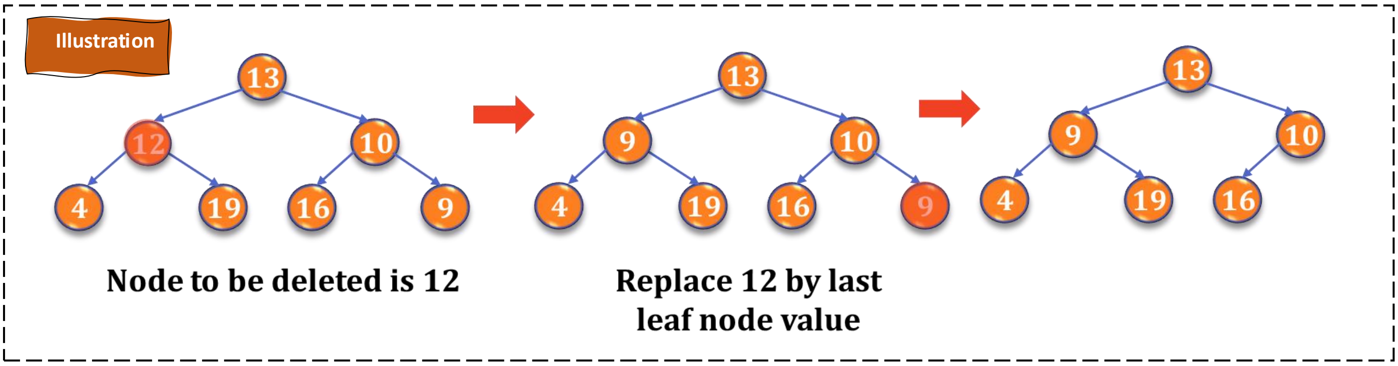
Binary Tree Insertion:

The key is inserted at the first position available in level order traversal.



Binary Tree Deletion:

The deleted node is replaced by the last leaf node.



✓ Binary Search Tree (BST)

A binary search tree (BST) is a binary tree that is empty or that satisfies the BST ordering property:

- The key of each node is greater than each key in the left subtree, if any, of the node.
- The key of each node is less than each key in the right subtree, if any, of the node.

- Each key in a BST is unique. (No duplication)

COMMON MISUNDERSTANDING!

☒ The key of each node is greater than the key of the left ***child*** of that node.

✓ The key of each node is greater than the keys of the left ***Subtree*** of that node.

BST Traversal & Properties:

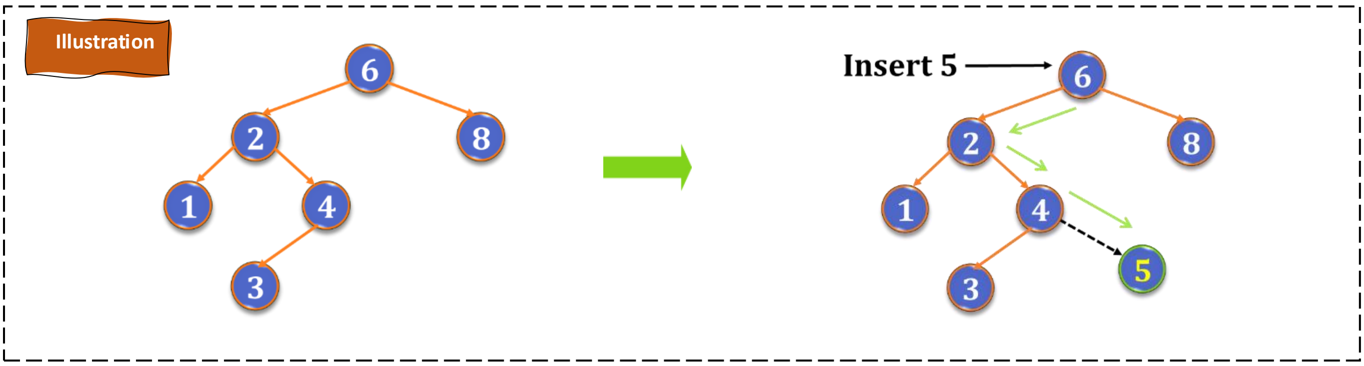
- Inorder traversal (L-N-R) of BST produces the output sorted in increasing order.
- Reverse Inorder traversal (R-N-L) of BST produces the output sorted in decreasing order.

BST (Search):

- Compare the key with tree nodes:
  - A match is found, or
  - If the key is less than the node, go left subtree and recheck.
  - If the key is greater than the node, go right subtree and recheck.
  - if the key is not found, throw an exception....
- Repeat this procedure until a match is found, or return false, null, or throw an exception.

BST (Insertion):

- By the BST ordering property, a new node is always inserted as a leaf node.
- The insert method recursively finds an appropriate empty subtree to insert the new key:
- If the key has already existed in the tree, an exception must be thrown. (No duplication in BST)



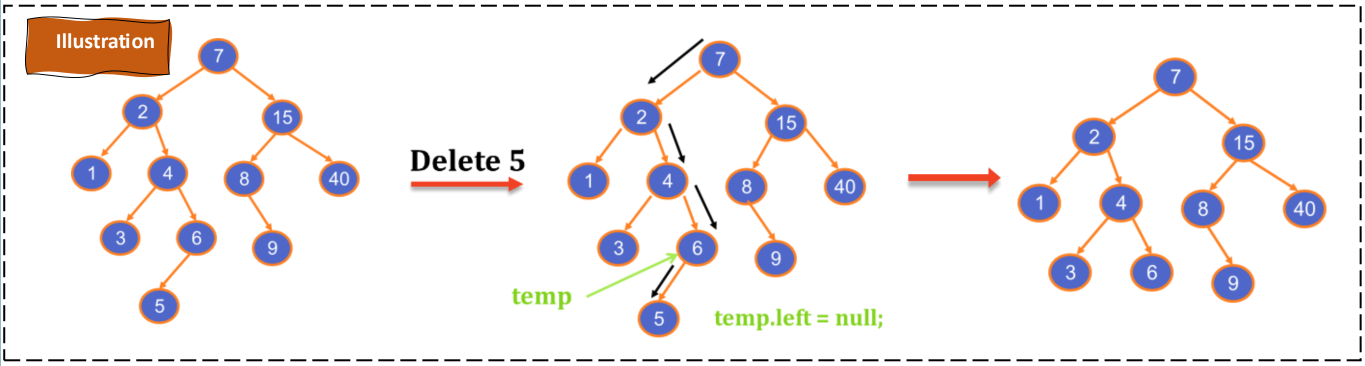
BST (Deletion):

There are three cases:

1) The node to be deleted is a leaf node.
2) The node to be deleted has one non-empty subtree.
3) The node to be deleted has two non-empty subtrees.

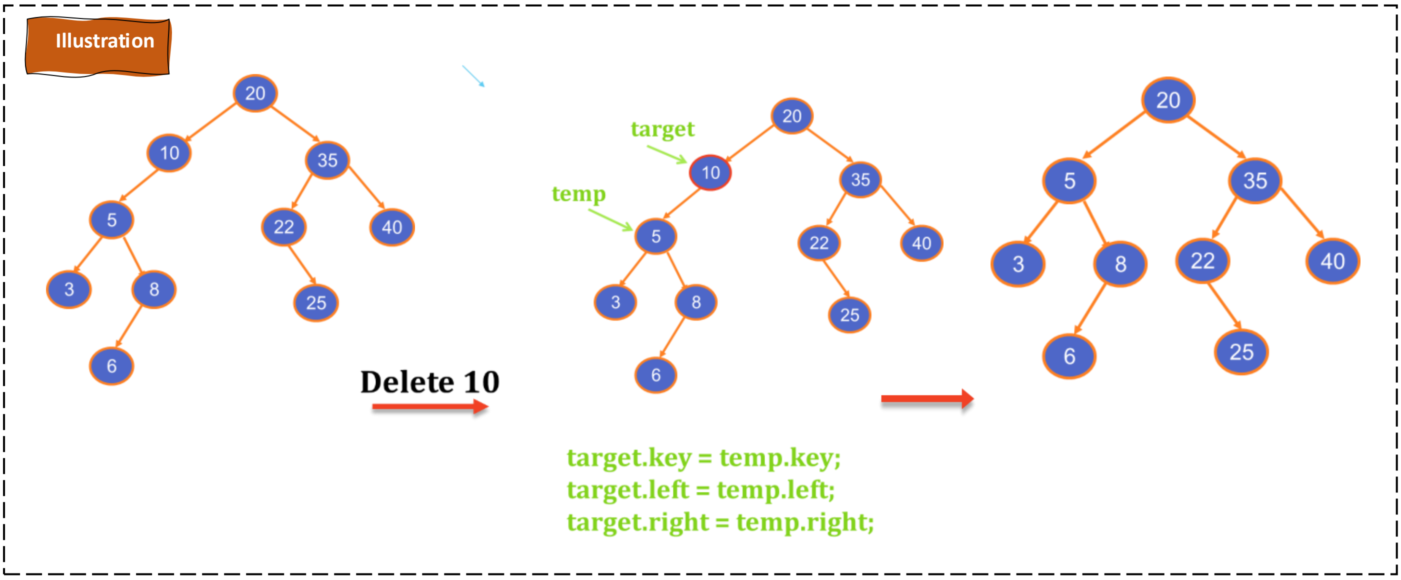
1) The node to be deleted is a leaf node:

- Delete the node by making the parent reference of that leaf node = null:



2) The node to be deleted has one non-empty subtree.

- Delete the node by replacing its value and references by the child value and references :



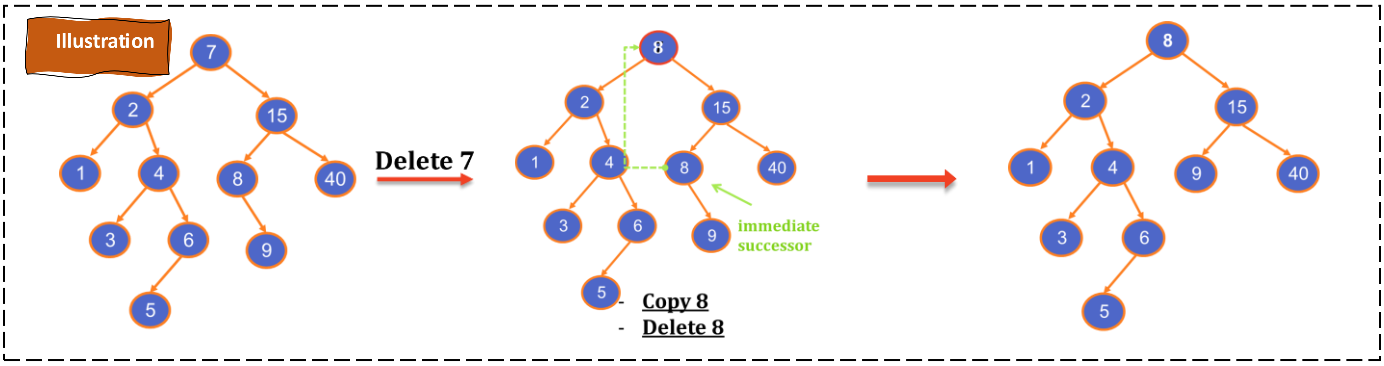
3) The node to be deleted **has two non-empty subtree.**

Two methods to delete node x:

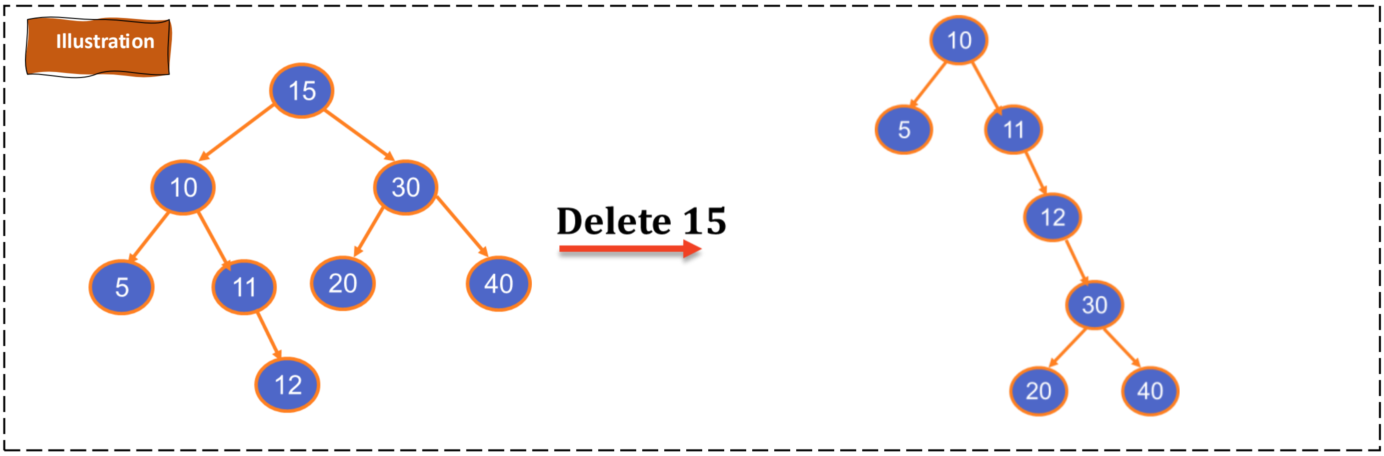
> **immediate successor:** The minimum key in the right subtree of x.
> **immediate predecessor:** The maximum key in the left subtree of x

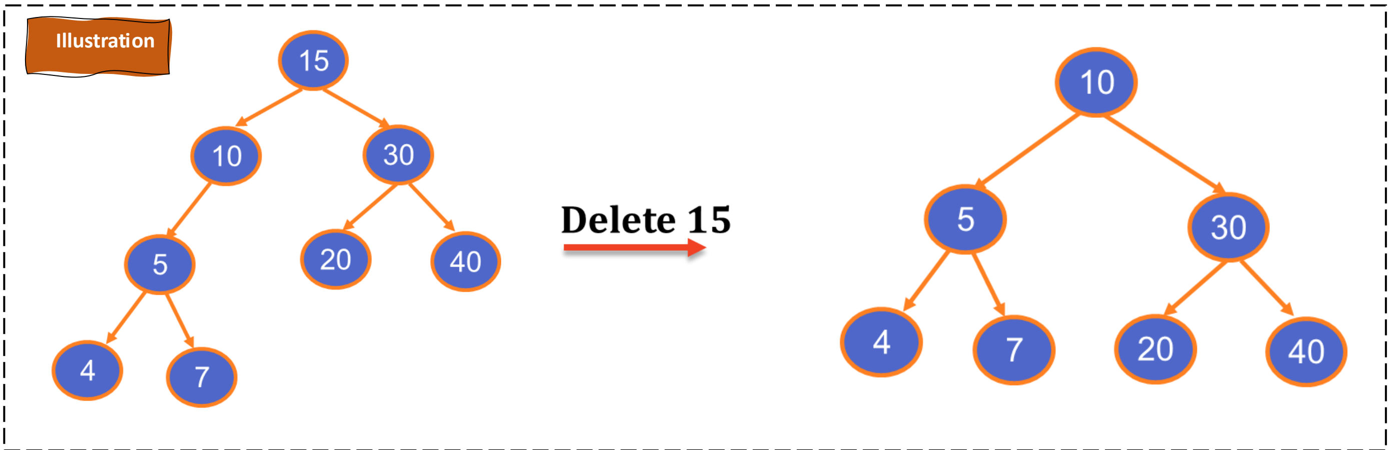
- **Method(1): Deletion by Copying:**
  - Replace the key being deleted with its immediate successor to the node x.
  - Delete that successor node.

(Same can be done by replacing the key being deleted with its immediate predecessor)



- **Method(2): Deletion by Merging:**
  - Taking one subtree out of the two subtrees of deleted node and then attaching it to the another subtree in a way that does not affect the BST property:





BSTs as Priority Queues:
By using insert and deleteMax or deleteMin, a BST can be used as a priority queue in which the keys of the elements are distinct.

Why BST?
BSTs provide good logarithmic time performance in the best and average cases.
Worst case is $O(n)$.

Operation	Retrieval or Search	Insertion	Deletion	Traversal
BST	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$

BSTs are used:

- **for indexing in databases.**
- **to implement various searching and sorting algorithms.**
- **to implement dictionaries, dynamic sets, and lookup tables.**

Average case complexities (linear data structures & BSTs):

Data Structure	Retrieval or Search	Insertion	Deletion	Traversal
BST	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$
Sorted Array	$O(\log n)$ (Using Binary Search Tree)	$O(n)$	$O(n)$	$O(n)$
Unsorted Array	$O(n)$	$O(1)$	$O(n)$	$O(n)$
Sorted Linked List	$O(n)$	$O(n)$	$O(n)$	$O(n)$
LinkedList	$O(n)$	$O(1)$	$O(n)$	$O(n)$

(BST worst execution time for each of the above operations is **O(n)** when the tree is linear (i.e., degenerate)).

Disadvantages of Binary Search Trees:

- The overall shape of the tree depends on the order of insertion. It might become linear or degenerate (Worst Scenario)
- The plain implementation of Binary Search Tree is not much used since it cannot guarantee logarithmic complexity.