Trees

Fall 2023 - Data Structures

Trees

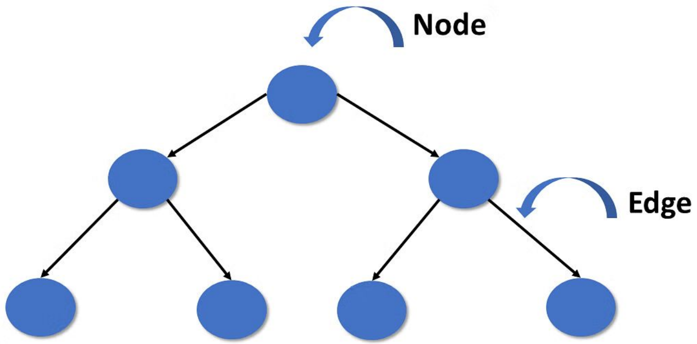
- Array is good for data which is accessed randomly and it is easy to implement.
- Linked Lists are ideal for applications which has frequent operations such as:
  - add, delete, update
- However, **searching** in these data structures are slow.
  - Because they are **linear**
  - Can we use something else?

Trees

- Do we have a data structure which has multiple relations among its nodes?
  - We call those a **tree**
- A tree is a collection of nodes connected by directed or undirected edges.
- It is a **nonlinear** data structure compared to arrays, linked lists, stacks and queues.
  - They are **linear**.

Trees

- Non-linear, hierarchical data structure
- Comprises a collection of entities known as **nodes**
- Connects each node in the data structure by using **edges**
    - Can be directed or undirected.



Why use it?

- Linear data structures store data in sequential order.
    - (This doesn’t mean they are sequentially stored in the memory)
- Time complexity increases when you want to perform operations.

Node

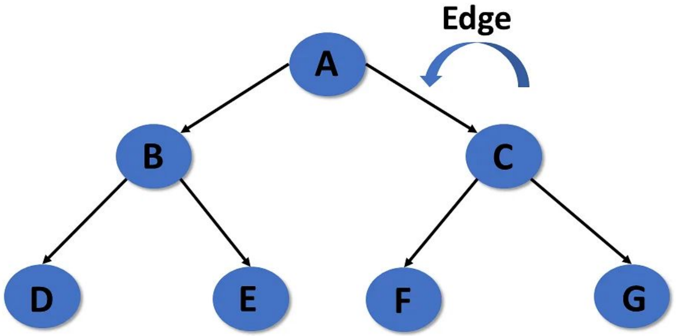
- A node is a structure that contains a key or value and pointers in its child node.
- Each node can have an arbitrary number of children and it is not known in advanced.
  - The general tree can be implemented using a **first child/next sibling** method.
  - Each node will have two points.
  - One to leftmost child, and one to rightmost sibling.

Root

- Root is the **first** node of the tree.
- It is the **initial node**.
- In a tree, there can only be **one** root node.

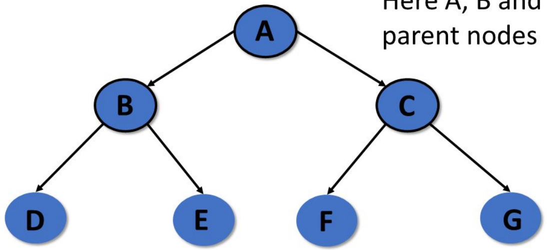
Edge

- The connecting link of any two nodes is called an **edge**.
- If there are **N** number of nodes, there are **N-1** edges.



Parent

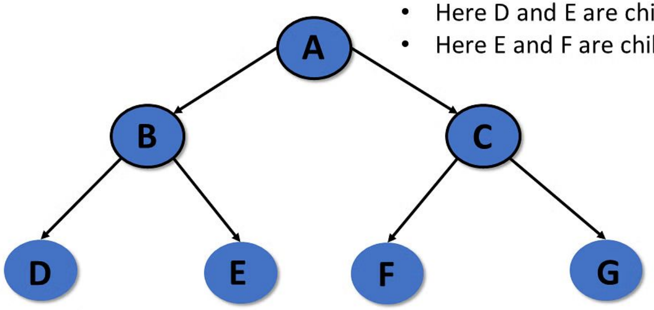
- A node which is predecessor of any node is known as a **parent** node.
- A node with a branch from itself to any other successive node is also called a **parent** node.



Child

- A descendant of any node is known as child node.
- Every node other than the **root** node is a **child** node.
    - Why?
- Any number of parent nodes can have any number of child nodes.

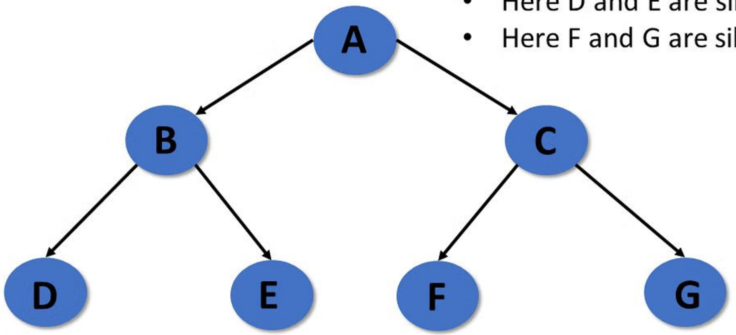
- Here B and C are children of A
- Here D and E are children of B
- Here E and F are children of C



Siblings

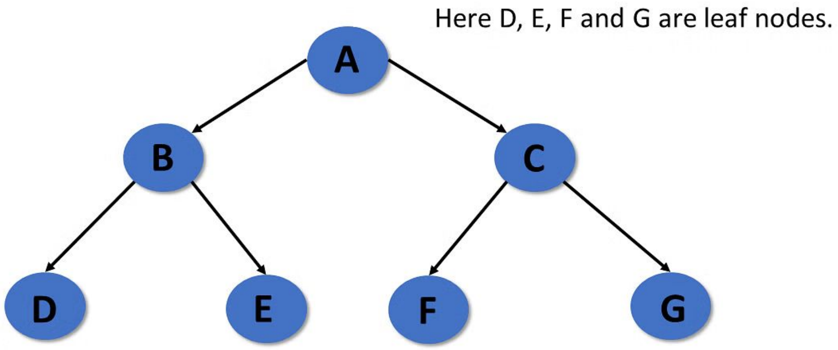
- Nodes that belong to the same parent are called **siblings**.

- Here B and C are siblings
- Here D and E are siblings
- Here F and G are siblings



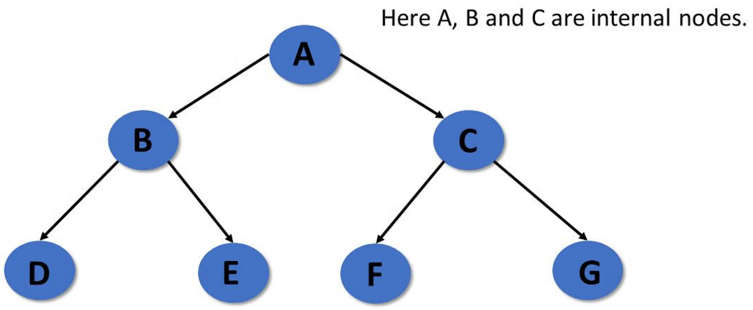
Leaf

- The node with no child is known as a **leaf** node.
- Also called **external nodes** or **terminal nodes**.
-



Internal nodes

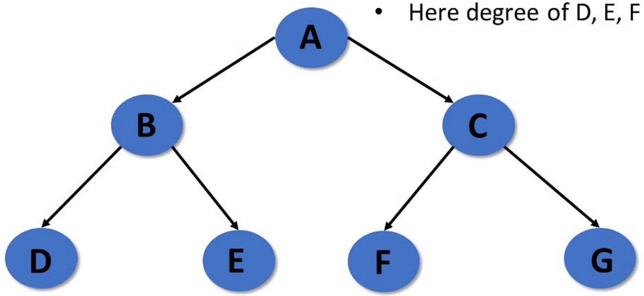
- Trees have at least one child node known as **internal nodes**
- Nodes other than leaf nodes (external nodes) are internal nodes.



Degree

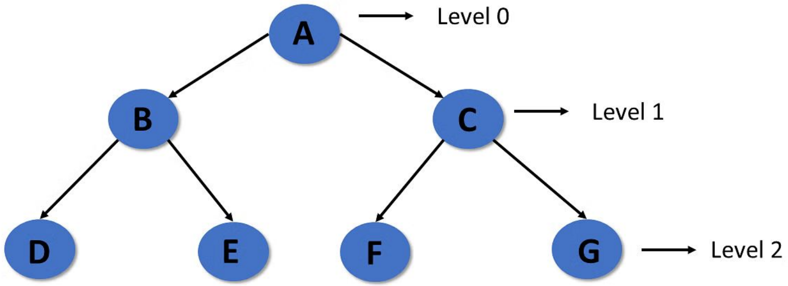
- Total number of children of a node is called the **degree** of the node.
- Highest degree of the node among all the nodes in a tree is the **Degree of Tree**

- Here degree of A, B and C is 2.
- Here degree of D, E, F and G is 0.



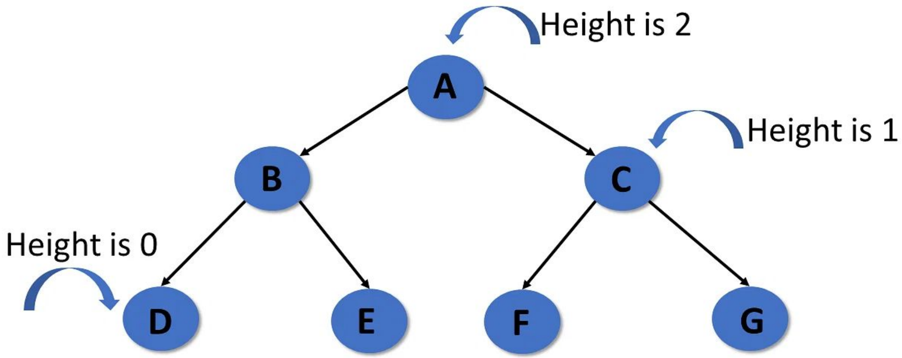
Level

- Root node is at level 0.
- Root node’s children are at level 1
- Children of them are at level 2
- …



Height

- Number of edges from the leaf node to the particular node in the longest path is known as the **height of that node**
- The height of **root** node is called **Height of Tree**
- Tree height of all **leaf** nodes are 0.



Depth

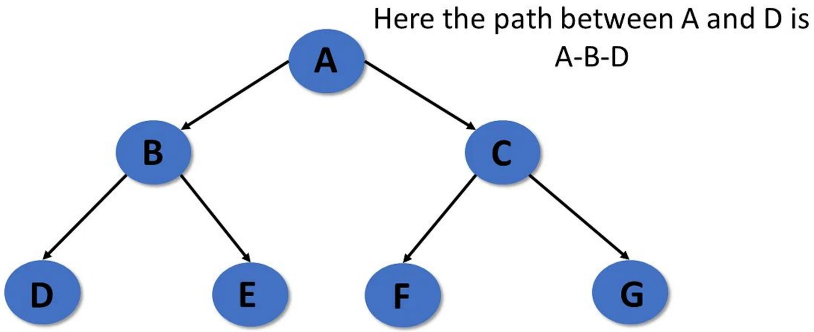
- Depth of a node refers to its distance from the root node.
  - Root node is depth 0
  - Any node connected directly to root is 1.
- Depth of a tree is the maximum level any node is situated at the tree.
  - Length of the longest path from the root node to the **furthest** leaf node.
  - Each step from a node to immediate child counts as 1.

Depth vs Level

- When we talk about a specific node, depth and level will be the same.
  - Try to find an example where they are not the same.
- However, they do not exactly mean the same thing.
- Depth:
  - Used when discussing the position of a single node
  - We say a tree hasa depth of 3
    - Means there are three **edges** from the root node to the deepest node.
- Level
  - Used in the context of all nodes that lie at the same distance from the root.
  - We say nodes at level 2
    - Refers to all nodes that are two steps away from the root, regardless of their individual path.

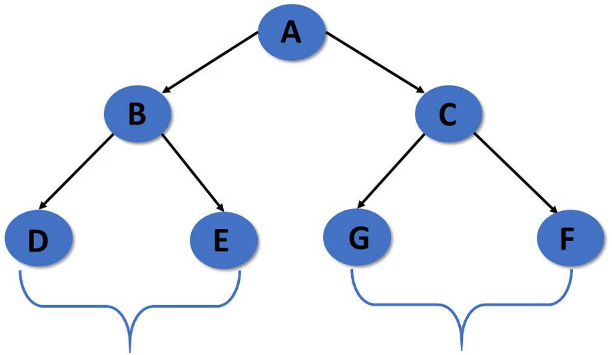
Path

- Sequence of nodes and edges from one node to another node is called the path between the two nodes.
- Length of a path is the total number of nodes in a path.



Types of Trees

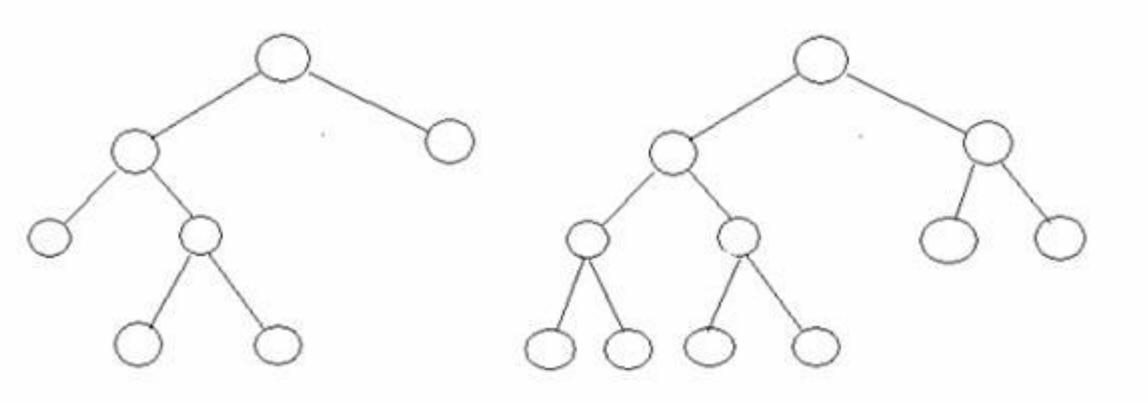
- General Tree
  - When there are no constraints on the hierarchical structure.
  - A node can have any number of nodes.
    - Which means a node can have any number of children they want.
- Binary Tree
  - A node can have at most two child nodes.
  - These are called **left child** and **right child**



A node can have any at most two children

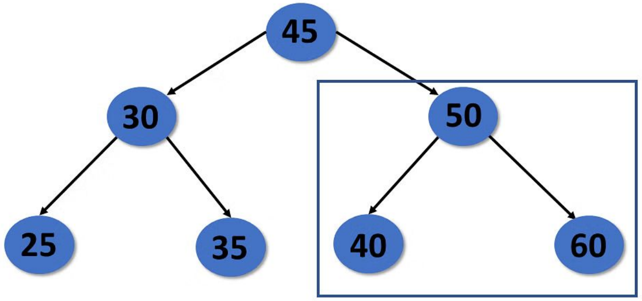
Binary Trees

- A binary tree in which each node has exactly **zero** or **two** children is called a **full binary tree (Left)**
  - There are no nodes with exactly one child.
- **A complete binary tree** is a tree which is completely filled. (Right)



Binary Search Tree (BST)

- More constricted extension of binary tree
- Has a unique property
  - Binary search property
  - Meaning: the value of a left child node should be **less than or equal to the parent node**
  - Value of right node should be **greater than or equal to the parent value**.



Left node value<= root node <= right node value

BST

- Binary tree != BST
- All BST are binary trees but not all binary trees are BST
- Key difference is how the nodes are organized
  - Binary tree is a more general concept with no specific ordering
  - BST is a specialized tree that maintains a specific order
    - To allow efficient searching and sorting operations
- Binary Trees are used in scenarios where data hierarchy is important but order is not.
  - They are the foundational structure of BST
- BST are used in scenarios where quick search, insertion and deletion are required. Useful for operations like lookup, addition and removal.
  - The special property of it allows us to skip half of the tree at each step.

BST Operations - Inserting a node

- Remember that BST is a special tree. In order for it to be BST, it needs to satisfy some conditions.
- Therefore, for every node **N**, all nodes in its **left** subtree have values less than **N**, all nodes in its right subtree have values **greater** than N.
- We want to add **X**.
  - Starting from the root, compare X with the value of the current node.
  - If **X** is smaller go left child. Else, go right child.
  - Repeat this process until you reach the leaf node and insert the value as left or right depending on its value.

BST Operations - Searching for a node

- Similar to insertion
- Start from the root and compare the value to be searched with the current node’s value.
- If they match, it means you’ve found the node.
- If the value is smaller, search in the left subtree, else in the **right**.
- Continue until the value is found **or** a leaf node is reached without finding the value.
    - It means that value is not in the tree.

BST Operations - Deleting a node

- More complex compared to other operations.
- You need to maintain the BST property after removal.
- Three cases to consider:
  - Node with No children:
    - Just remove the node
  - Node with One Child
    - Remove the node and replace it with its child
  - Node with Two Children
    - Find the node’s in-order successor or in-order predecessor.
    - Replace the node’s value with the in-order successor/predecessor value
    - Delete the in-order successor/predecessor.
    - **In-order successor:** The smallest node in its right subtree
    - **In-order Predecessor:** The largest node in its left subtree

Usage

- Often used in databases for indexing.
- Allows for quick search, insertion and deletion of records.
- Used in autocomplete features
- File system and Hierarchical Structure Representation
  - File systems can be represented as such. Pre or post-order can be used for operations like displaying directory contents, calculating folder sizes, etc.
- Expression tree evaluation
  - Used to represent arithmetic expressions.
  - Post-order traversal is used to evaluate these expressions. (Reverse Polish notation - [4 5 +])
- etc.

Tree traversal

- Traversal of the tree is the process of visiting each node and printing their value.
- Traversing a linked list or an array is easy, but in BST; not so much.
  - Traversals can also be applied to regular trees.
- Three ways:
  - Pre-order traversal
    - Root, Left, Right
    - Useful for creating a copy of the tree
  - In-order traversal
    - Left, Root, Right
    - Retrieves elements in sorted order
  - Post-order traversal
    - Left, Right, Root
    - Useful for deleting the tree

In-order traversal

- The idea is that we visit the nodes in the order **left-root-right**
- **Hint:**
  - It should always be sorted!
  - In-order traversal **for BST** gives us the sorted result in ascending order (increasing)

Pre-Order Traversal

- Root - Left - Right
- Processed as the node is visited.
- Primary Use
  - Visits the root before the subtrees, making it useful for operations where you need to visit ancestors before descendants.
- Applications
  - Copying a tree, expressing it in a way that it can be reconstructed (serialization), prefix notation in expression trees, creating a prefix expression (Polish Notation), and tree traversals in graph algorithms.

Post-Order Traversal

- Left - Right - Root
- Node is not processed until the children are.
- Primary Use
  - Root is visited last after its subtrees, making it suitable for post-processing subtrees before processing the parent.
- Applications
  - Useful in deleting or freeing nodes and subtrees, postfix expression evaluation (Reverse Polish Notation), solving certain dynamic programming problems, etc.

InOrder(root) visits nodes in the following order:

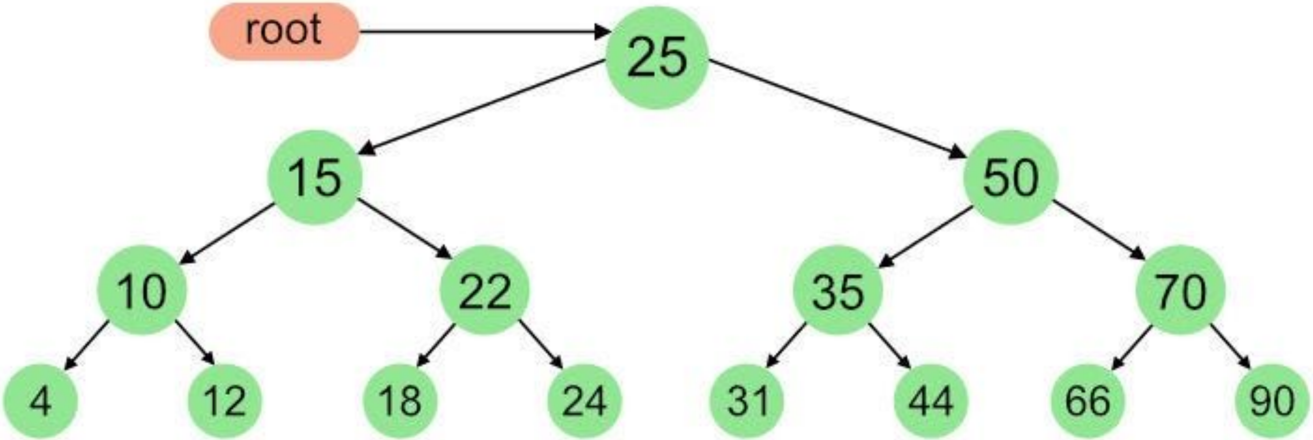
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:

25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

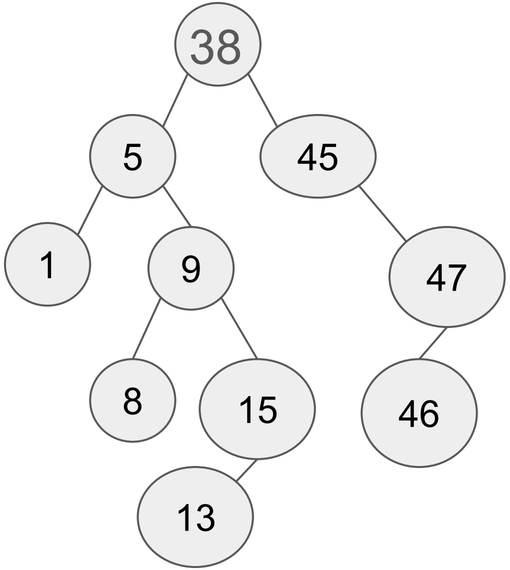
A Post-order traversal visits nodes in the following order:

4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



Example

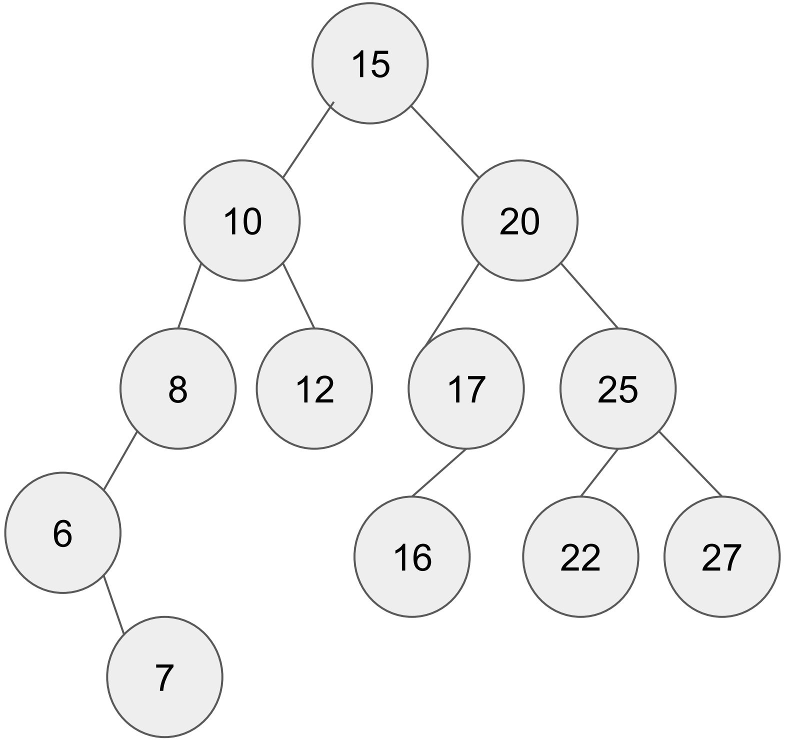
- Write it in array form for 3 traversals



Solution

- In-order: 1, 5, 8, 0, 13, 15, 38, 45, 46
- Pre-order: 38, 5, 1, 9, 8, 15, 13, 45, 47, 46
- Post-order: 1, 8, 13, 15, 9, 5, 46, 45, 38

Exercise



Solution

- In-order
- 6, 7, 8, 10, 12, 15, 16, 17, 20, 22, 25, 27
- Remember, in-order is sorted order.

Solution

- Pre-order
- 15 10 8 7 6 12 20 17 16 25 22 27

Solution

- Post-order traversal
- 7 6 8 12 10 16 17 22 27 25 20 15

Example

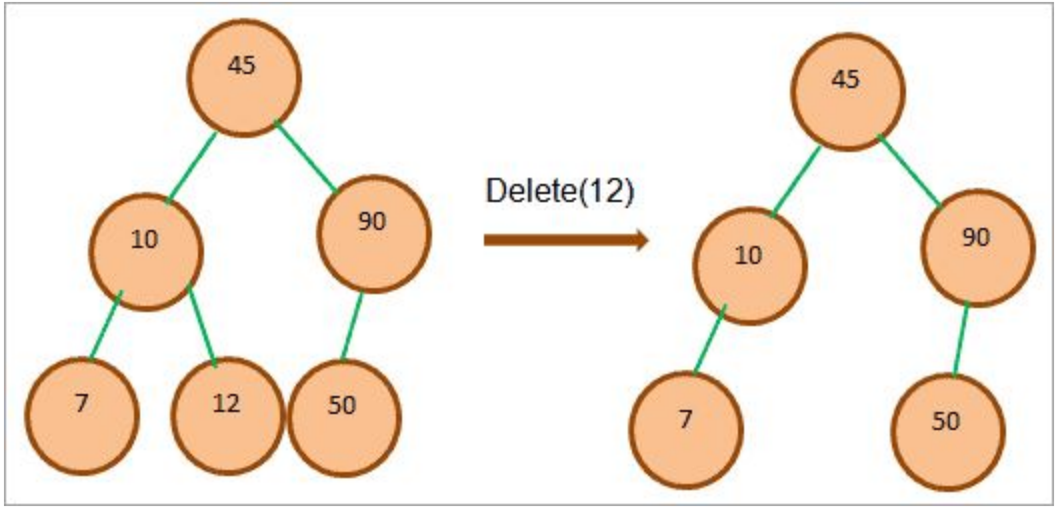
- Array: 45, 10, 7, 90, 12 ,50, 13, 39, 57
- Draw a BST
- At every element, check for the elements before. If it is smaller go left, if larger go right.

Example

- Add another number to the previous BST after creating it.

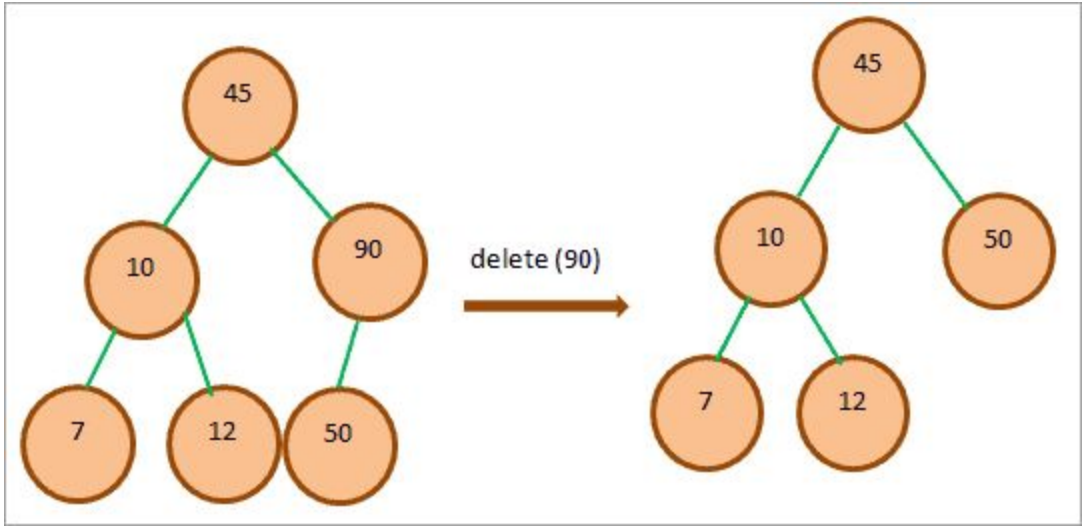
Example (delete leaf node)

- Delete 12
- Since it is a leaf node we can delete it.



Example (delete one child)

- Delete 90
- There is a single child of 90.
- We copy the value
- Delete 90
- Paste copied there



Example (Node with 2 children)

- Delete 45
- If right subtree is not empty, replace root with minimum node in right subtree.