Data Structures
Topic: Trees

Definition

- A **tree** is a nonlinear **data structure**
- Tree is a hierarchical data structure which stores the information naturally in the form of hierarchy style
- It represents the nodes connected by edges.

Field	Description
Root	Root is a special node in a tree. The entire tree is referenced through it. It does not have a parent.
Parent Node	Parent node is an immediate predecessor of a node.
Child Node	All immediate successors of a node are its children.
Siblings	Nodes with the same parent are called Siblings.
Path	Path is a number of successive edges from source node to destination node.
Height of Node	Height of a node represents the number of edges on the longest path between that node and a leaf.
Height of Tree	Height of tree represents the height of its root node.
Depth of Node	Depth of a node represents the number of edges from the tree's root node to the node.
Degree of Node	Degree of a node represents a number of children of a node.
Edge	Edge is a connection between one node to another. It is a line between two nodes or a node and a leaf.

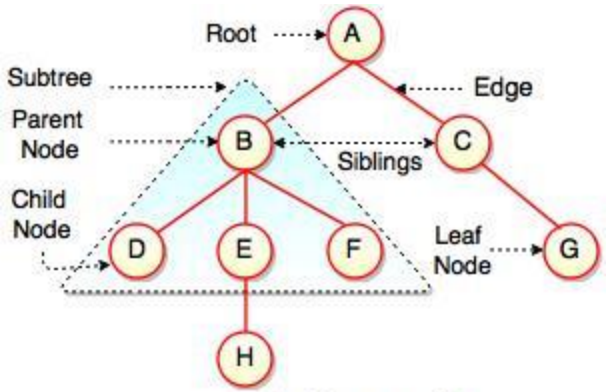


Fig. Structure of Tree

Levels of a node

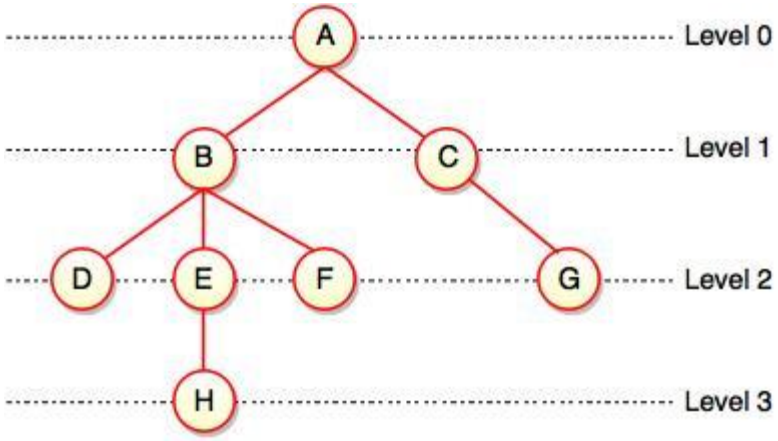


Fig. Levels of Tree

If node has no children, it is called **Leaves** or **External Nodes.**

- Nodes which are not leaves, are called **Internal Nodes**. Internal nodes have at least one child.

Height of a Node

Height of a node defines the longest path from the node to a leaf.

- Path can only be downward.

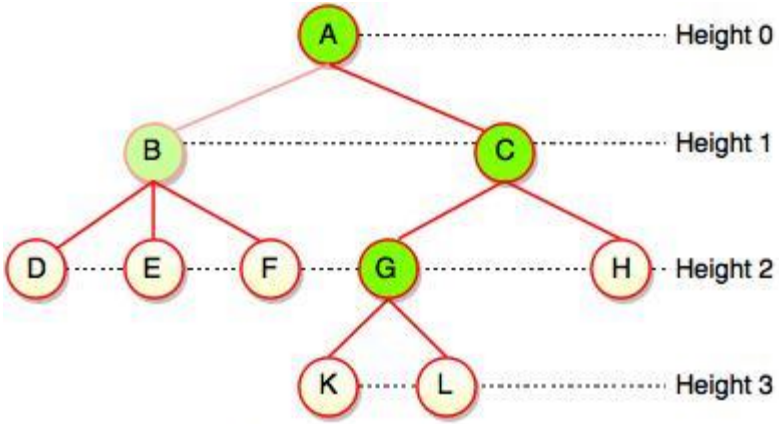


Fig. Height of a Node

Node A's height is the number of edges of the path to K not to D. And its height is 3.

Depth of a Node

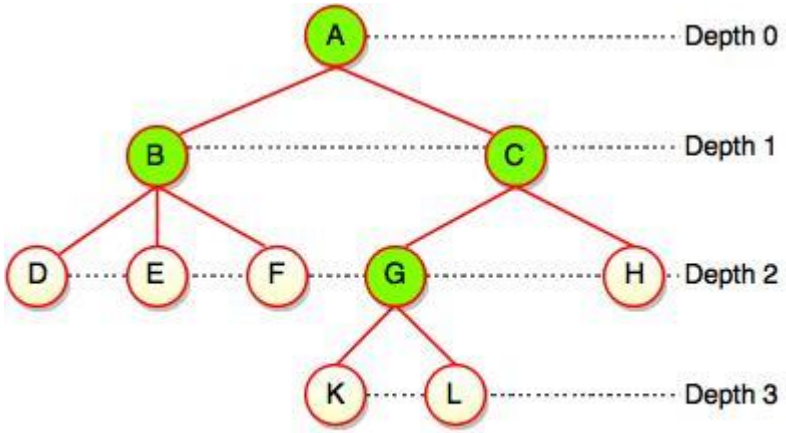


Fig. Depth of a Node

While talking about the height, it locates a node at bottom where for depth, it is located at top which is root level and therefore we call it depth of a node.

In the above figure, Node G's depth is 2. In depth of a node, we just count how many edges between the targeting node & the root and ignoring the directions.

Note: Depth of the root is 0.

Advantages of Tree

- Tree reflects structural relationships in the data.
- It is used to represent hierarchies.
- It provides an efficient insertion and searching operations.
- Trees are flexible. It allows to move subtrees around with minimum effort.

Applications

- Routing Tables
- Decision Trees
- Expression Evaluation
- Java Objects (JVM)
- Indices for Databases
- Facebook, Instagram
- Google Maps
- World Wide Web
- Load balancing and interrupt handling in OS
- Caches and garbage collection
- Rabin-karp Algorithm- plagiarism
- Game boards

Binary Tree

- A tree whose elements have at most 2 children is called a binary tree.
- Since each element in a binary tree can have only 2 children, we typically name them the left and right child.

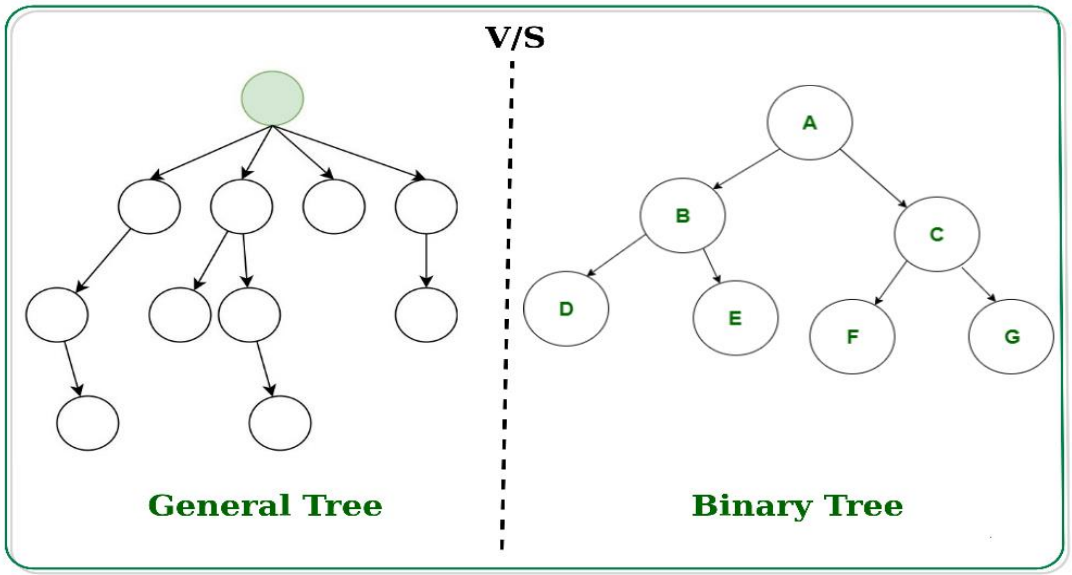


Fig 1: General Tree vs Binary tree

Contd..

- **Root:** The root of a tree is the topmost node of the tree that has no parent node. There is only one root node in every tree.
- **Edge:** Edge acts as a link between the parent node and the child node.
- **Leaf:** A node that has no child is known as the leaf node. It is the last node of the tree. There can be multiple leaf nodes in a tree.
- **Subtree:** The subtree of a node is the tree considering that particular node as the root node.
- **Depth(level):** The depth of the node is the distance from the root node to that particular node.
- **Height:** The height of the node is the distance from that node to the deepest node of that subtree.
- **Height of tree:** The Height of the tree is the maximum height of any node. This is same as the height of root node

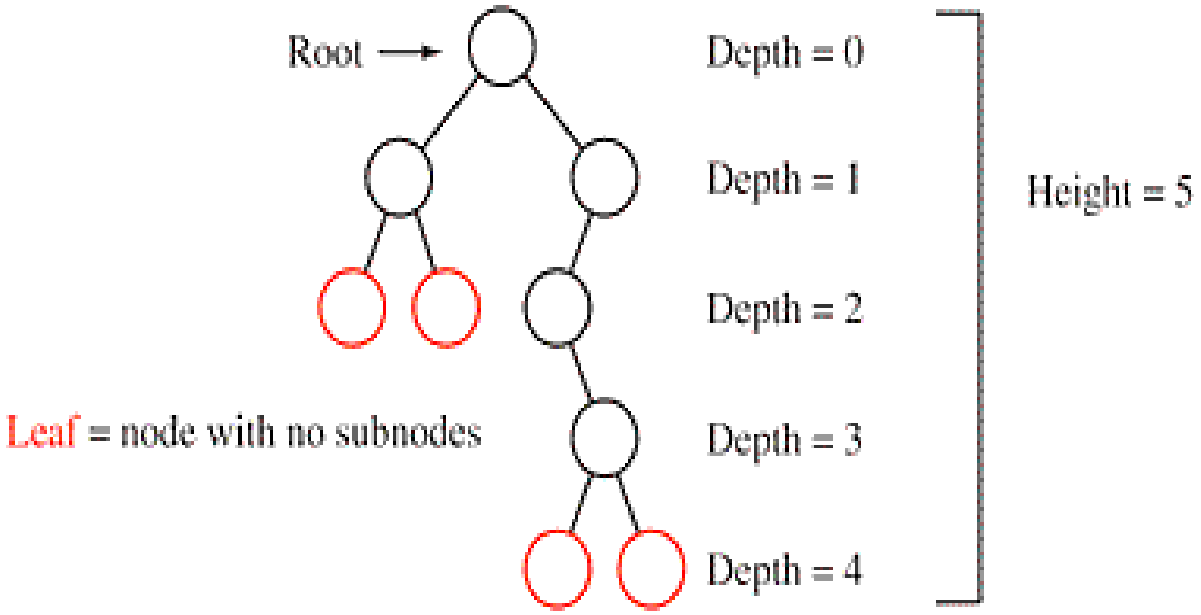


Fig 2: Understanding Depth and Height

Properties of Binary Tree

The maximum number of nodes at level 'l' of a binary tree is 2^l.

Here level is the number of nodes on the path from the root to the node (including root and node).

Level of the root is 0.

This can be proved by induction.

For root, l = 0, number of nodes = $2^0 = 1$

Assume that the maximum number of nodes on level 'l' is 2^l

Since in Binary tree every node has at most 2 children, next level would have twice nodes, i.e. $2 * 2^l$

Contd..

The Maximum number of nodes in a binary tree of height 'h' is $2^h - 1$

Here the height of a tree is the maximum number of nodes on the root to leaf path.

Height of a tree with a single node is considered as 1.
A tree has maximum nodes if all levels have maximum nodes.

So maximum number of nodes in a binary tree of height h is

$1 + 2 + 4 + .. + 2^{h-1}$.

This is a simple geometric series with h terms and sum of this series is $2^h - 1$.
In some books, the height of the root is considered as 0. In this convention, the above formula becomes $2^{h+1} - 1$

Contd..

In a Binary Tree with N nodes, minimum possible height or the minimum number of levels is $Log_2(N+1)$.

- There should be at least one element on each level, so the height cannot be more than N.
- A binary tree of height 'h' can have maximum $2^h - 1$ nodes (previous property).
- So the number of nodes will be less than or equal to this maximum value.

Contd..

$N <= 2^h - 1$

$2^h >= N+1$

$\log_2(2^h) >= \log_2(N+1)$ (Taking log both sides)

$h\log_2 2 >= \log_2(N+1)$ (h is an integer)

$h >= \lceil \log_2(N+1) \rceil$

So the minimum height possible is $\lceil \log_2(N+1) \rceil$

Contd..

- In Binary tree where every node has 0 or 2 children, the number of leaf nodes is always one more than nodes with two children.

- L = T + 1
- Where L = Number of leaf nodes
- T = Number of internal nodes with two children

Types of binary Trees

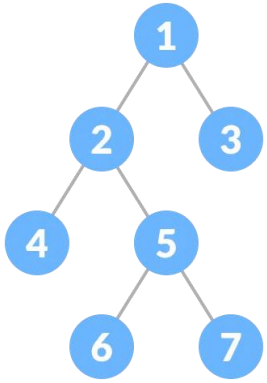
Strict Binary Tree: A full Binary tree is a special type of binary tree in which every parent node/internal node has either two or no children.

Full Binary Tree: A perfect binary tree is a type of binary tree in which every internal node has exactly two child nodes and all the leaf nodes are at the **same level**.

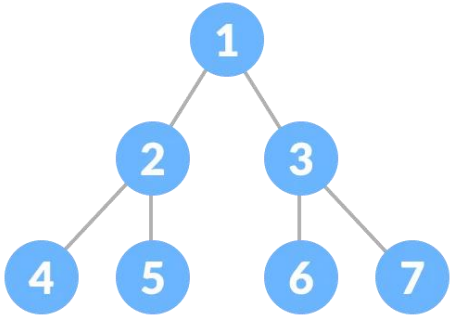
A complete binary tree: It is a binary tree in which all the levels are completely filled except possibly the lowest one, which is filled from the left.

Left skewed Binary Tree: These are those skewed binary trees in which all the nodes are having a left child or no child at all. It is a left side dominated tree. All the right children remain as null.

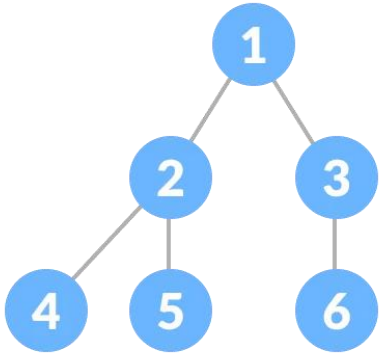
Right Skewed Binary Tree**:** These are those skewed binary trees in which all the nodes are having a right child or no child at all. It is a right side dominated tree. All the left children remain as null.



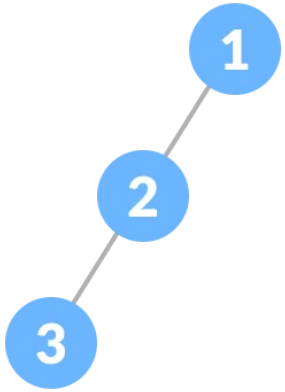
a. Strict binary tree



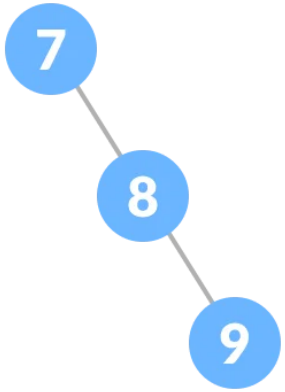
b. Full binary tree



c. Complete binary tree



d. Left skewed BT



e. Right skewed BT

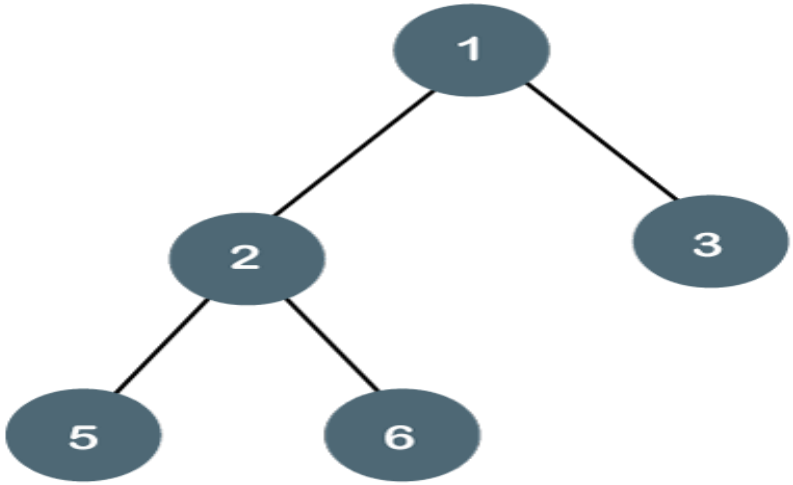
Fig 3: Types of Binary trees

Tree Representations

Array:

In array representation of a binary tree, we use one-dimensional array (1-D Array) to represent a binary tree.

The array representation stores the tree data by scanning elements using level order fashion. So it stores nodes level by level. If some element is missing, it left blank spaces for it.



1	2	3	4	5	6	7
1	2	3	5	6	-	-

Contd..

Linked List Representation

- We use a double linked list to represent a binary tree.
- In a double linked list, every node consists of three fields.
- First field for storing left child address, second for storing actual data and third for storing right child address.

```
struct node
{
   int data;
   struct node* left;
   struct node* right;
};
```

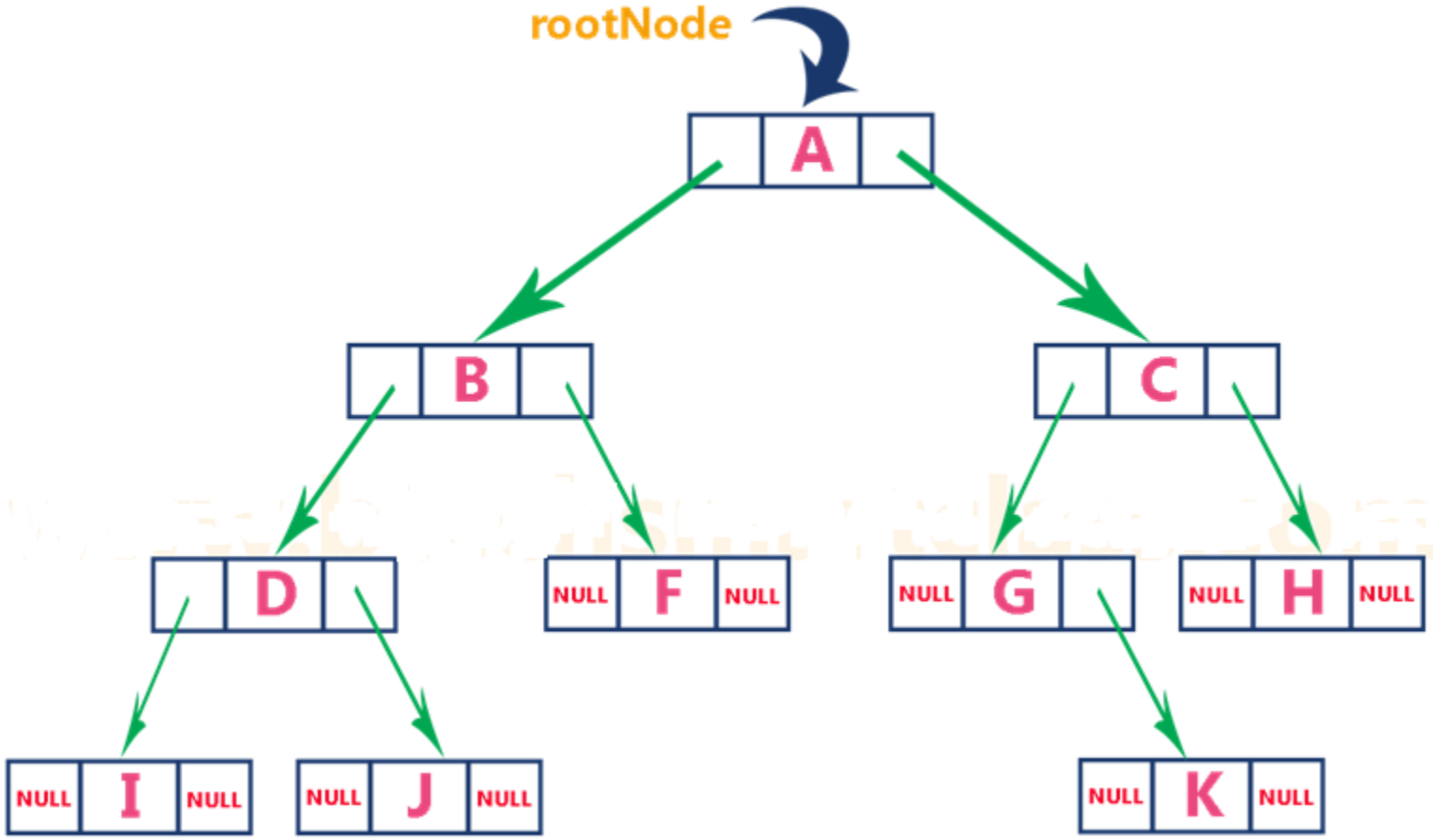


Fig 4: Linkedlist representation

Traversals

- Depth First Traversal:
  - PreOrder Traversal: root – left child – right child.
  - InOrder Traversal: left child – root – right child.
  - PostOrder Traversal: left child – right child – root.
- Breadth First Traversal: Level by level.

Preorder

- In the preorder traversal, the root node is processed first, followed by the left subtree and then the right subtree.
- It draws its name from the Latin prefix pre, which means to go before.
- Thus, the root goes before the subtrees.

Contd..

Algorithm preOrder (root)
//Traverse a binary tree in node-left-right sequence.
//Pre root is the entry node of a tree or subtree

```
1: if (root is not null)
        1: process (root)
        2: preOrder (leftSubtree)
        3: preOrder (rightSubtree)
2: end if
end preOrder
```

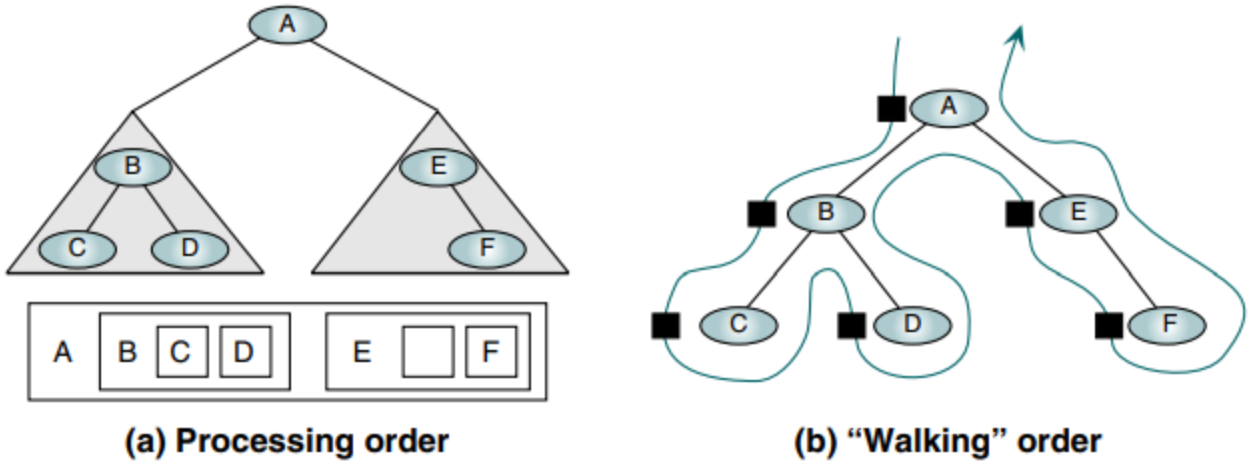


Fig 5: Preorder: A B C D E F

Contd..

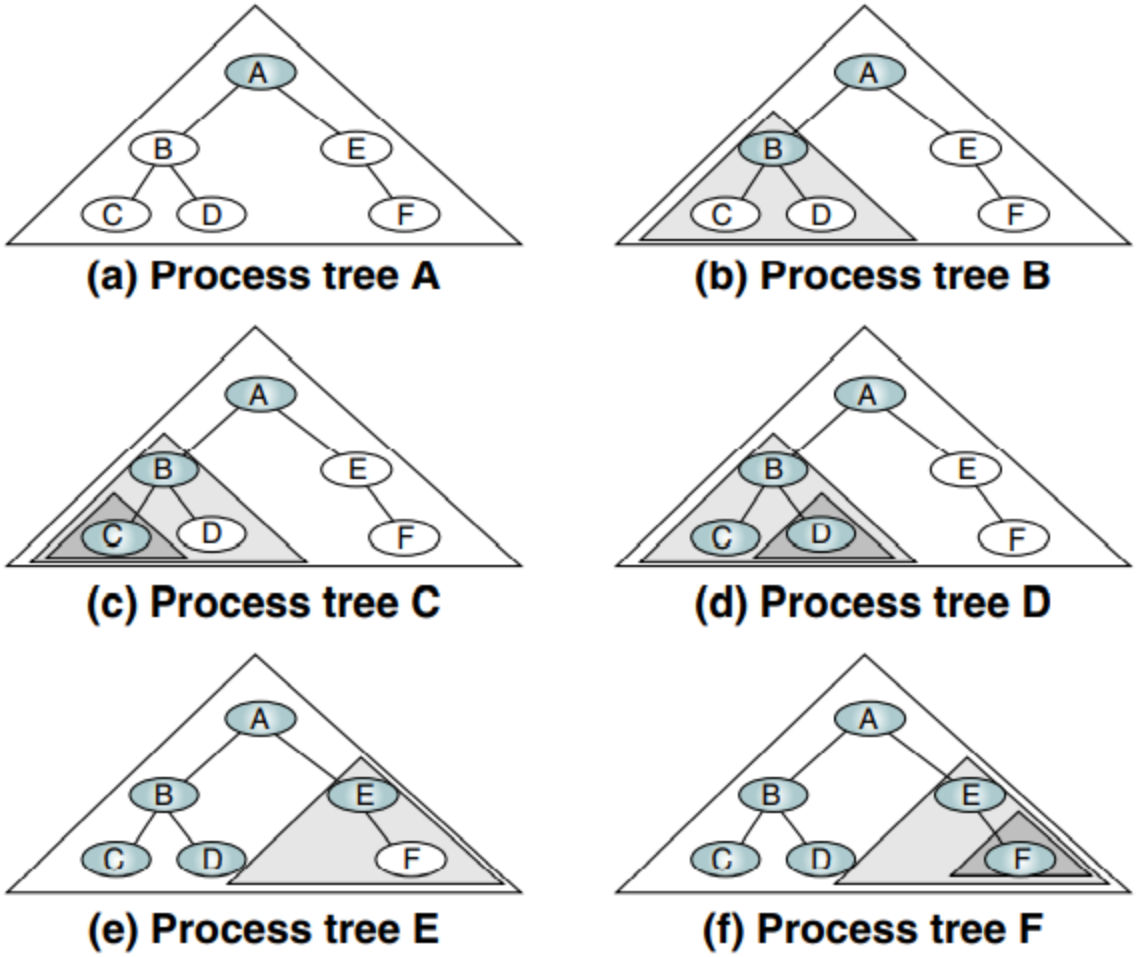


Fig 6: Algorithmic Traversal of Binary Tree

Inorder

- The inorder traversal processes the left subtree first, then the root, and finally the right subtree.

- The meaning of the prefix in is that the root is processed in between the subtrees.

Contd..

```
Algorithm inOrder (root)
//Traverse a binary tree in left-node-right sequence.
//Pre root is the entry node of a tree or subtree
1 if (root is not null)
        1 inOrder (leftSubTree)
        2 process (root)
        3 inOrder (rightSubTree)
2 end if
end inOrder
```

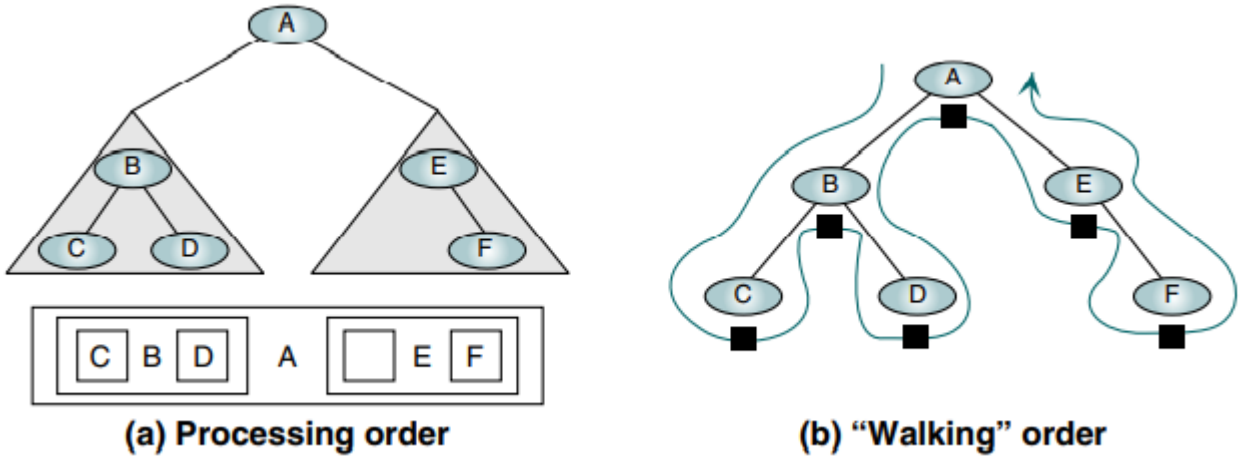


Fig 7: Inorder Traversal—C B D A E F

Postorder Traversal

- It processes the root node after (post) the left and right subtrees have been processed.

Contd..

```
Algorithm postOrder (root)
//Traverse a binary tree in left-right-node sequence.
//Pre root is the entry node of a tree or subtree
1 if (root is not null)
        1 postOrder (left subtree)
        2 postOrder (right subtree)
        3 process (root)
2 end if
end postOrder
```

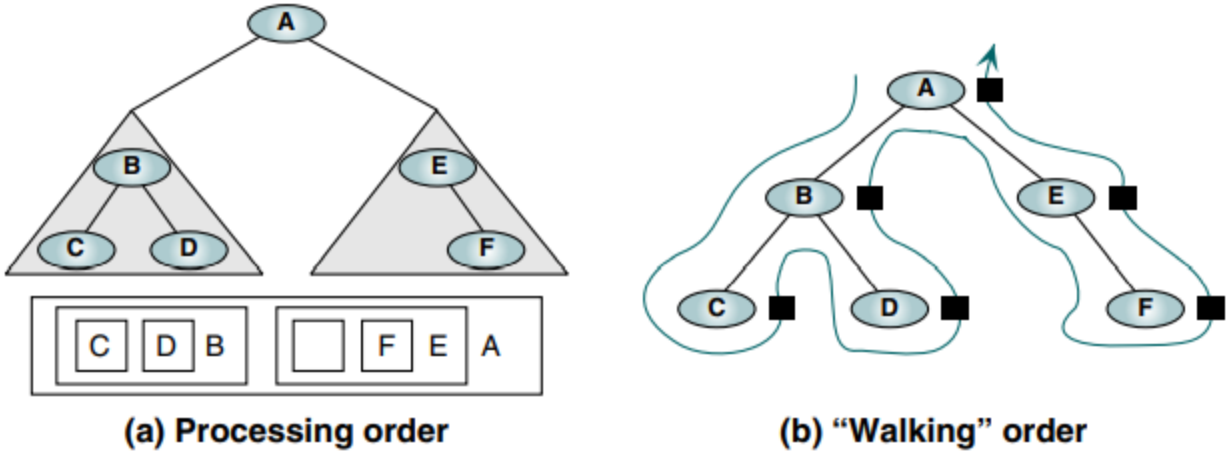


Fig 8: Postorder Traversal—C D B F E A

Breadth-first Traversals

- Level by Level order.
- In the breadth-first traversal of a binary tree, we process all of the children of a node before proceeding with the next level.
- In other words, given a root at level n, we process all nodes at level n before proceeding with the nodes at n+1.

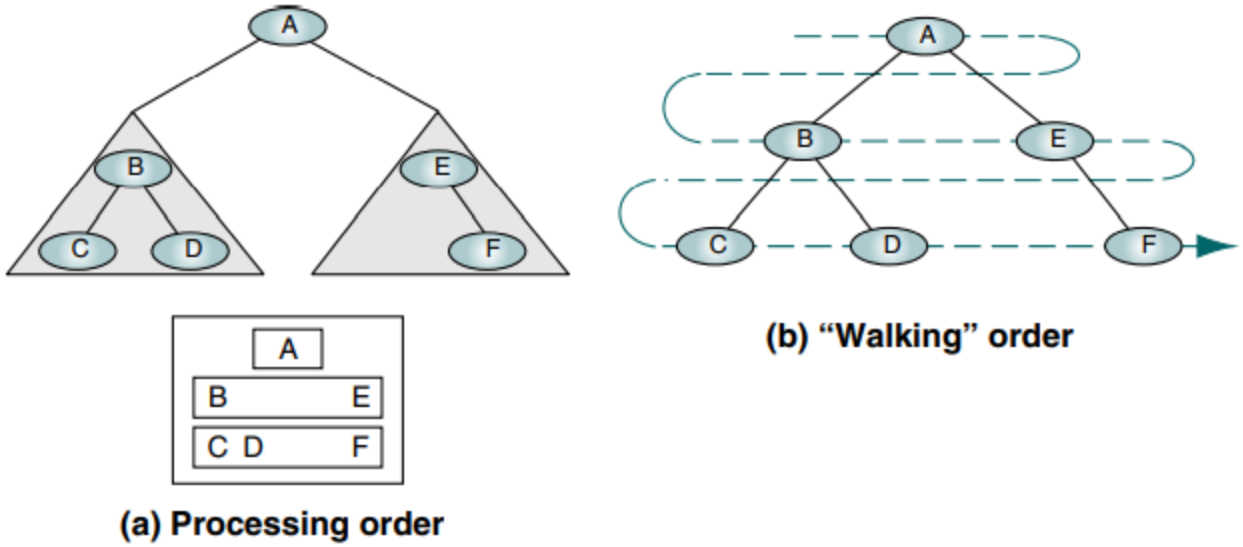
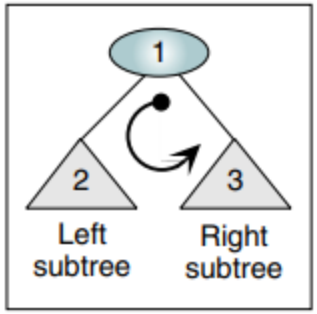
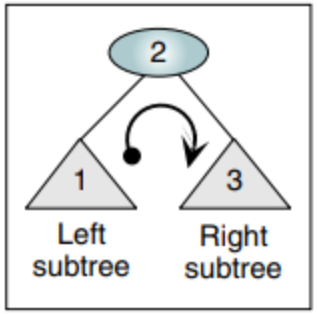


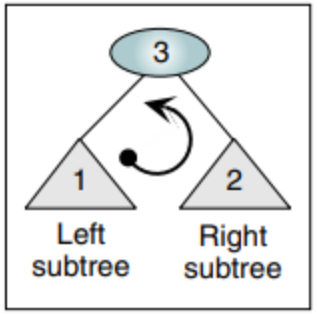
Fig 9: Breadth-first Traversal : A B E C D F



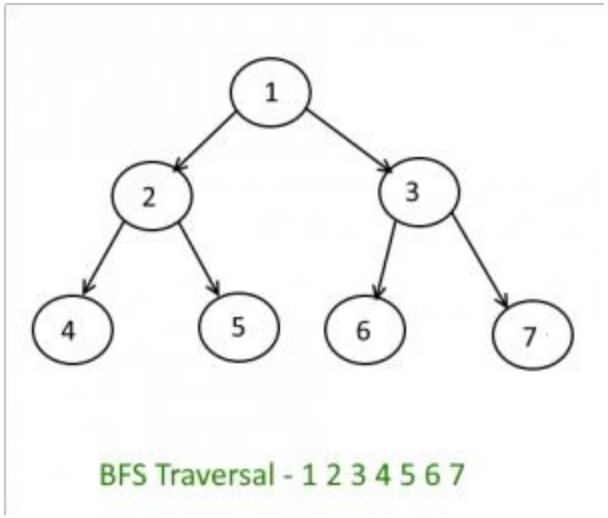
(a) Preorder traversal



(b) Inorder traversal



(c) Postorder traversal



Tree Traversal visualization links

Inorder : https://www.youtube.com/watch?v=4_UDUj1j1KQ

Preorder: https://www.youtube.com/watch?v=8xue-ZBlTKQ

Postorder: https://www.youtube.com/watch?v=4Xo-GtBiQN0

Construction of tree using traversals

- Pre requisite to construct binary tree are
  - Inorder and preorder
  - Inorder and postorder

Applications of Binary Trees

- Huffman coding tree is an application of binary trees that are used in data compression algorithms.
- In compilers, Expression Trees are used which are applications of binary trees.
- Represent hierarchical data.
- Used in editing software like Microsoft Excel and spreadsheets.
- Useful for indexing segmented at the database is useful in storing cache in the system,
- Syntax trees are used for most famous compilers for programming like GCC, and AOCL to perform arithmetic operations.
- For implementing priority queues.
- Used to enable fast memory allocation in computers.
- To perform encoding and decoding operations.

Advantages of Binary Tree:

- The searching operation in a binary tree is very fast.
- The representation of a binary tree is simple and easy to understand.
- Traversing from a parent node to its child node and vice-versa is efficiently done.
- Simple to implement
- Easy to understand.
- A hierarchical structure.
- Reflect structural relationships that are present in the data set
- Easy to insert data than in another data store.
- Easy to store data in memory management.
- User can many nodes
- Executions are fast.
- Store an arbitrary number of data values.

Disadvantages of Binary Tree:

- In binary tree traversals, there are many pointers that are null and hence useless.
- The access operation in a Binary Search Tree (BST) is slower than in an array.
- A basic option is dependent on the height of the tree.
- Deletion node not easy.
- A basic option is based on the height of tree.

Ordered Binary Trees: Binary Search Tree

Definition

- A binary search tree (BST) is a binary tree with the following properties:
  - All items in the left subtree are less than the root.
  - All items in the right subtree are greater than the root.
  - Each subtree is itself a binary search tree.

- Note: There must be no duplicate nodes.

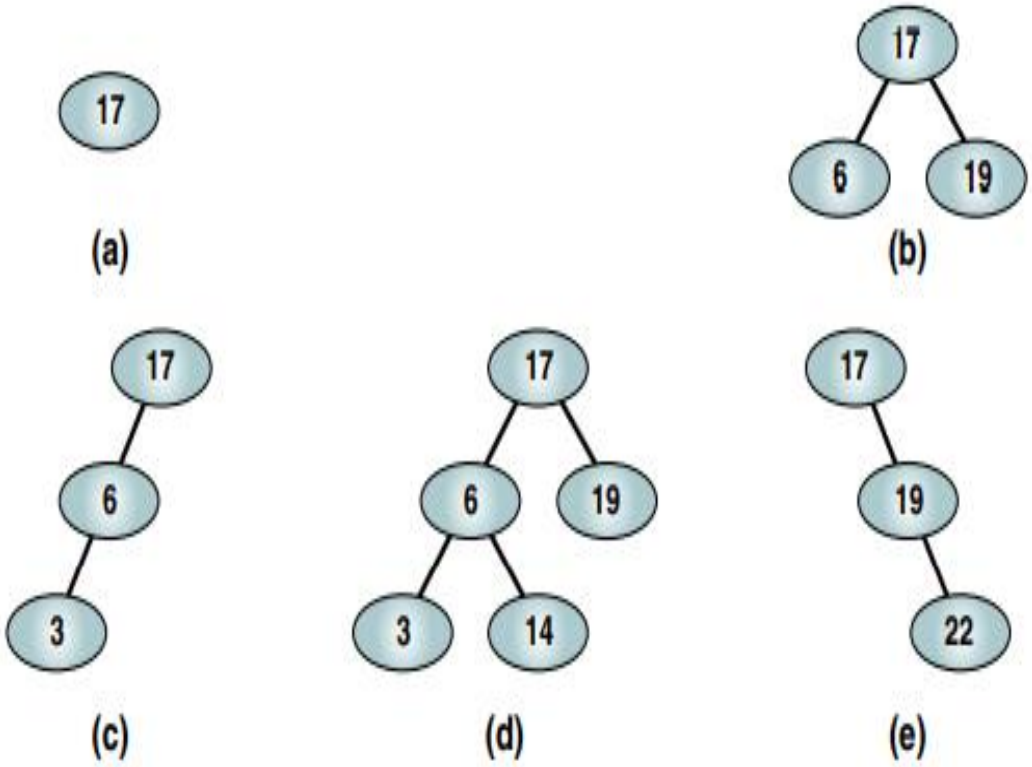


Fig 11 : Valid Binary Search Trees

Operations

- Insert
- Delete
- Search

Insert

- This is a very straight forward operation.
- If the tree is empty, then consider the first element as root.
- If the element is greater than root, it is added to the right subtree, and if it is lesser than the root, it is added to the left subtree.

Contd..

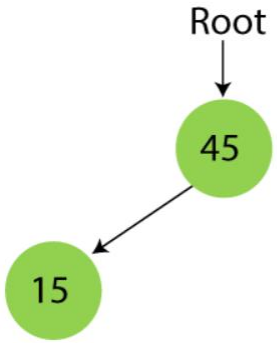
- Insert 45,15,79,90,10,55

Initially BST is empty so first element 45 is root.

Step 1 - Insert 45.

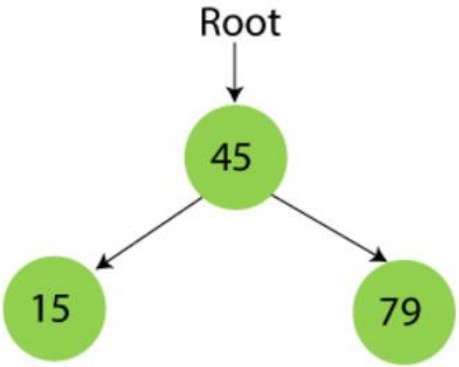
Step 2 - Insert 15.

As 15 is smaller than 45, so insert it as the root node of the left subtree.



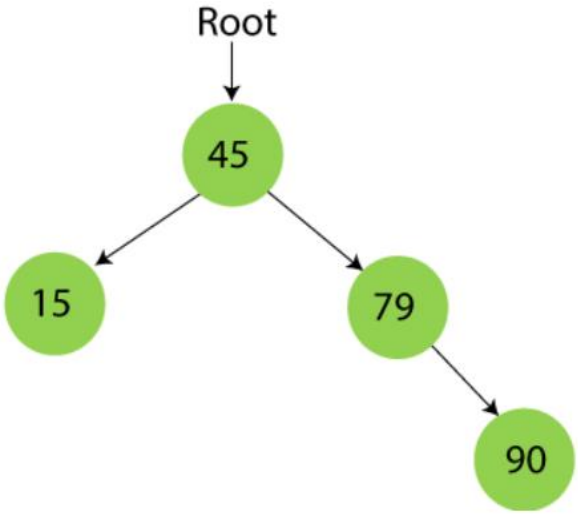
Step 3 - Insert 79.

As 79 is greater than 45, so insert it as the root node of the right subtree.



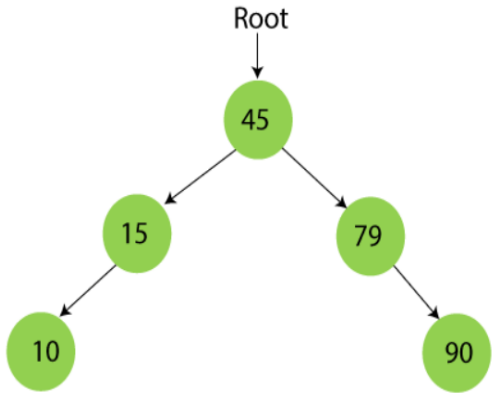
Step 4 - Insert 90.

90 is greater than 45 and 79, so it will be inserted as the right subtree of 79.



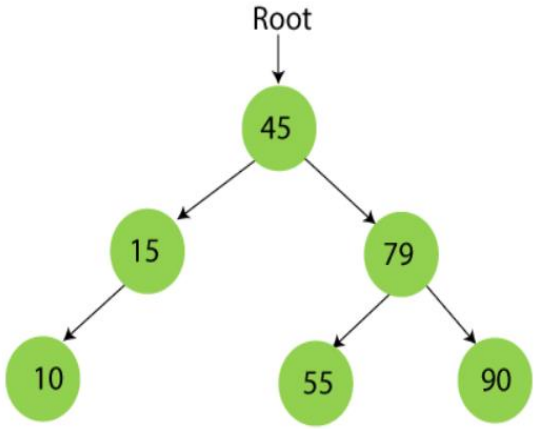
Step 5 - Insert 10.

10 is smaller than 45 and 15, so it will be inserted as a left subtree of 15.



Step 6 - Insert 55.

55 is larger than 45 and smaller than 79, so it will be inserted as the left subtree of 79



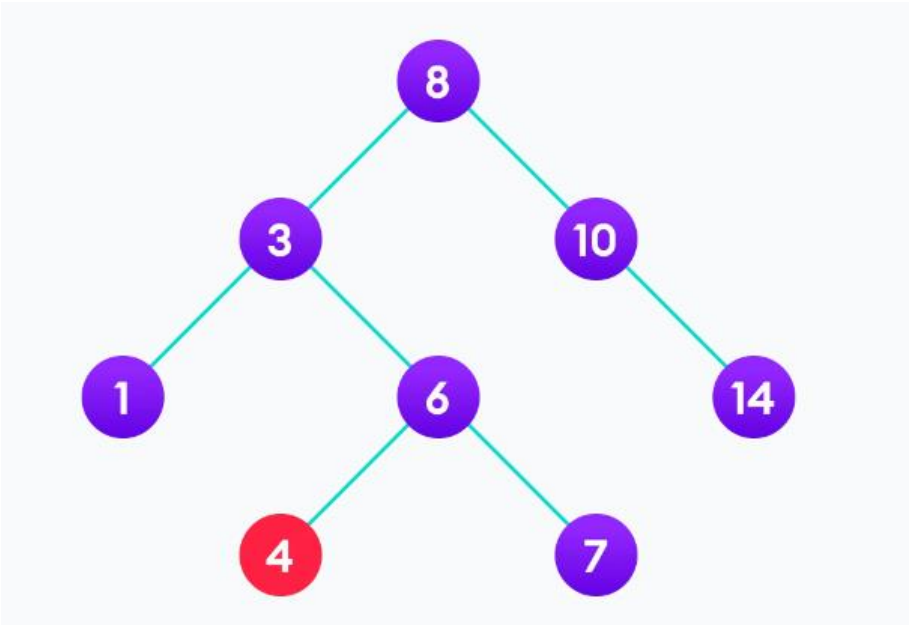
```
Algorithm addBST (root, newNode)
//Insert node containing new data into BST using recursion.
//Pre root is address of current node in a BST
//newNode is address of node containing data
//Post newNode inserted into the tree
//Return address of potential new tree root
1 if (empty tree)
        1 set root to newNode
        2 return newNode
2 end if
//Locate null subtree for insertion
3 if (newNode < root)
        1 return addBST (left subtree, newNode)
4 else
        1 return addBST (right subtree, newNode)
5 end if
end addBST
```

Deletion

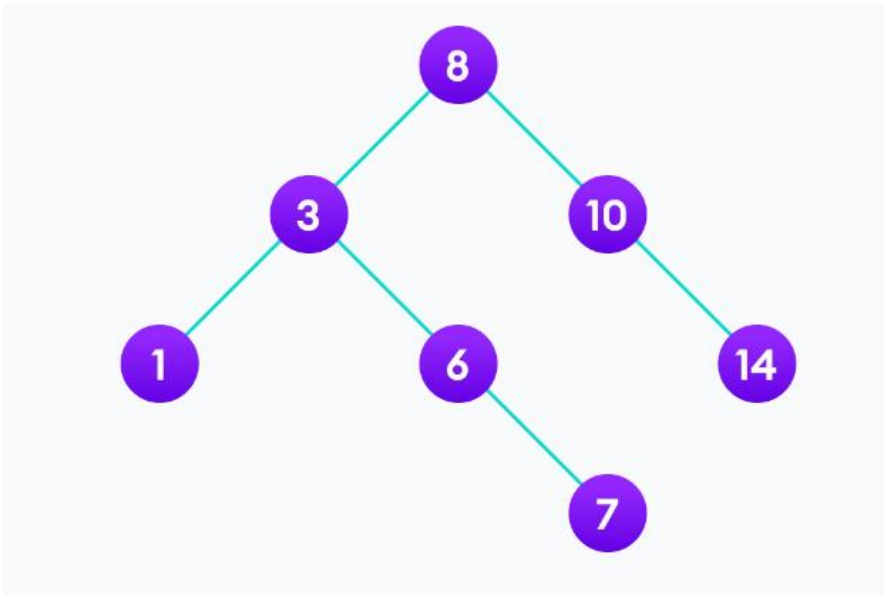
- In a binary search tree, we must delete a node from the tree by keeping in mind that the property of BST is not violated. To delete a node from BST, there are three possible situations occur :
  - The node to be deleted is the leaf node.
  - The node to be deleted has only one child.
  - The node to be deleted has two children.

Case I

In the first case, the node to be deleted is the leaf node. In such a case, simply delete the node from the tree.



Node 4 is to be deleted

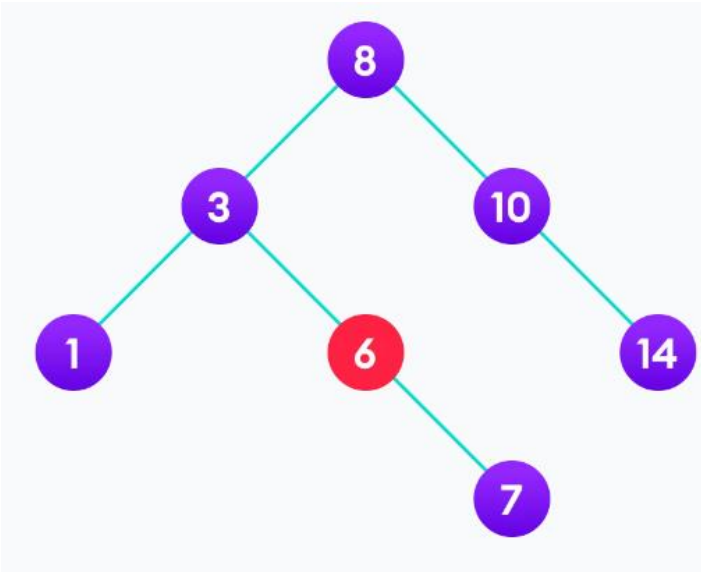


After deletion

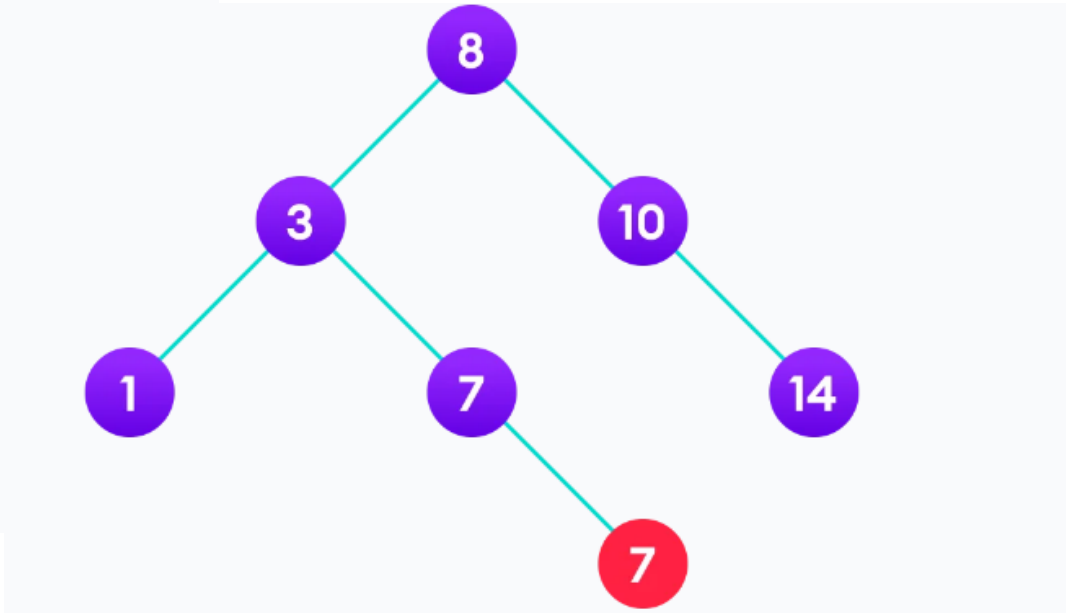
Case II

In the second case, the node to be deleted lies has a single child node. In such a case follow the steps below:

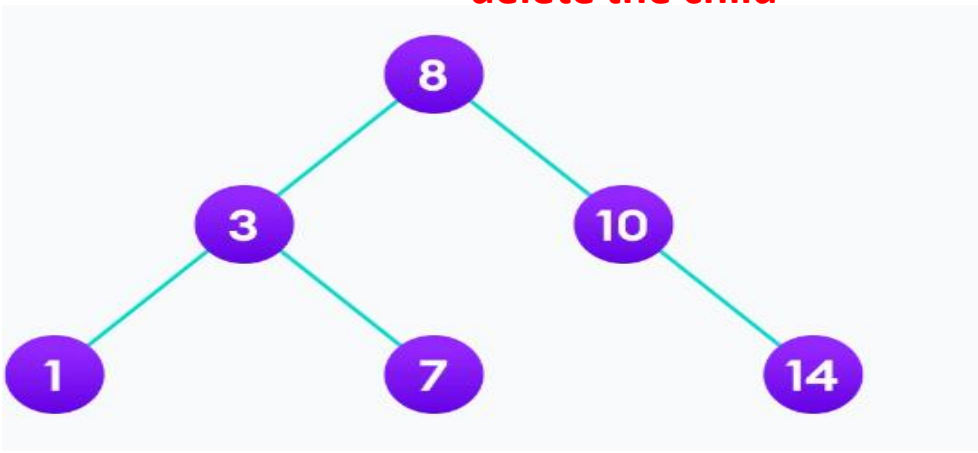
- Replace that node with its child node.
- Remove the child node from its original position.



6 is to be deleted



copy the value of its child to the node and delete the child

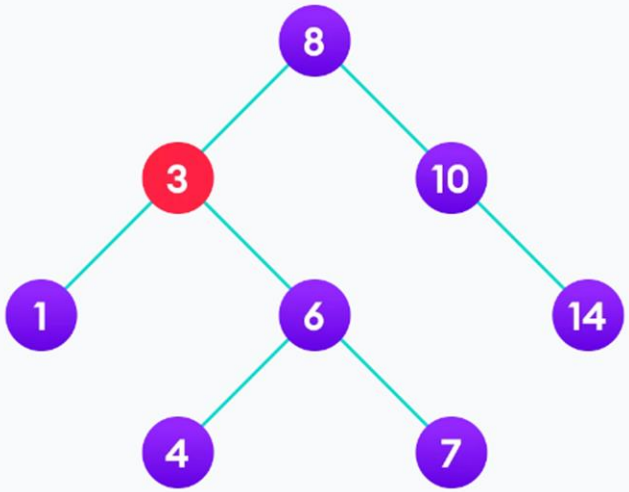


Final Tree

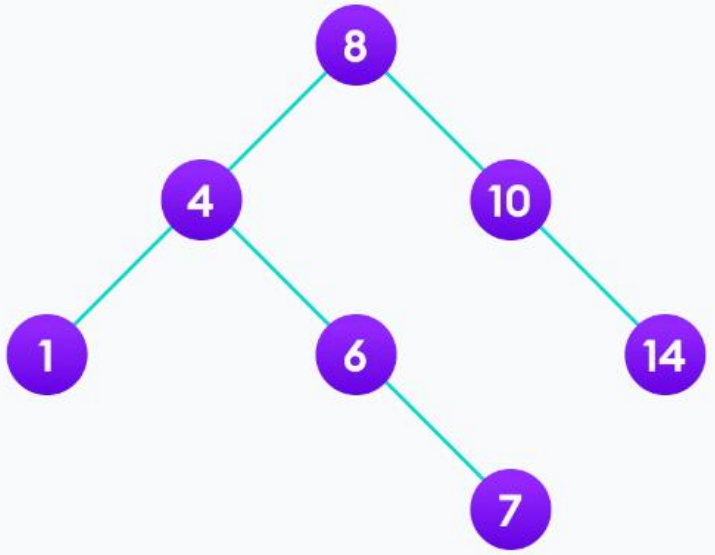
Case III

In the third case, the node to be deleted has two children. In such a case follow the steps below:

- Get the inorder successor of that node.
- Replace the node with the inorder successor.
- Remove the inorder successor from its original position.



3 is to be deleted



Copy the value of the inorder successor (4) to the node and Delete the inorder successor

Inorder:
1,3,4,6,7,8,10,14

Search

- Search operation is straightforward in a BST.
- Start with the root and keep moving left or right using the BST property.
- If the data we are searching is same as nodes data then we return current node.
- If the data we are searching is less than nodes data then search left subtree of current node;
- otherwise search right subtree of current node. If the data is not present, we end up in a NULL link.

Binary Search Tree Complexities

Time Complexity

Operation	Best Case Complexity	Average Case Complexity	Worst Case Complexity
Search	O(log n)	O(log n)	O(n)
Insertion	O(log n)	O(log n)	O(n)
Deletion	O(log n)	O(log n)	O(n)

Here, n is the number of nodes in the tree.

Space Complexity

The space complexity for all the operations is O(n).

Applications of BST

- BSTs are used for indexing in databases.
- It is used to implement searching algorithms.
- BSTs are used to implement Huffman coding algorithm.
- It is also used to implement dictionaries.

Advantages of Binary Search Tree:

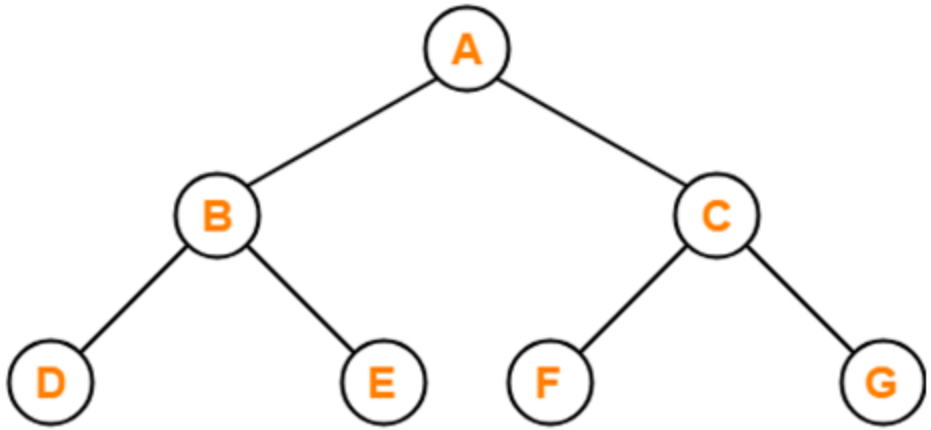
- BST is fast in insertion and deletion when balanced.
- BST is efficient.
- We can also do range queries – find keys between N and M (N <= M).
- BST code is simple as compared to other data structures.

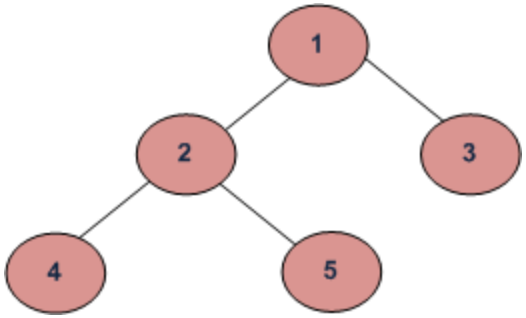
Disadvantages of Binary Search Tree:

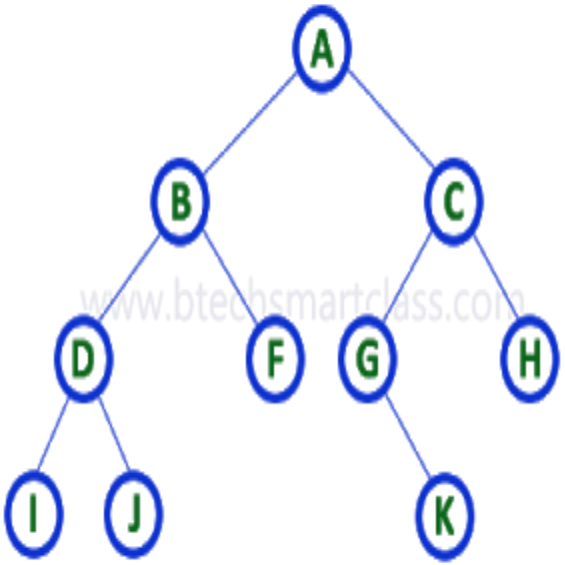
- The main disadvantage is that we should always implement a balanced binary search tree. Otherwise the cost of operations may not be logarithmic and degenerate into a linear search on an array.
- Accessing the element in BST is slightly slower than array.
- A BST can be imbalanced or degenerated which can increase the complexity

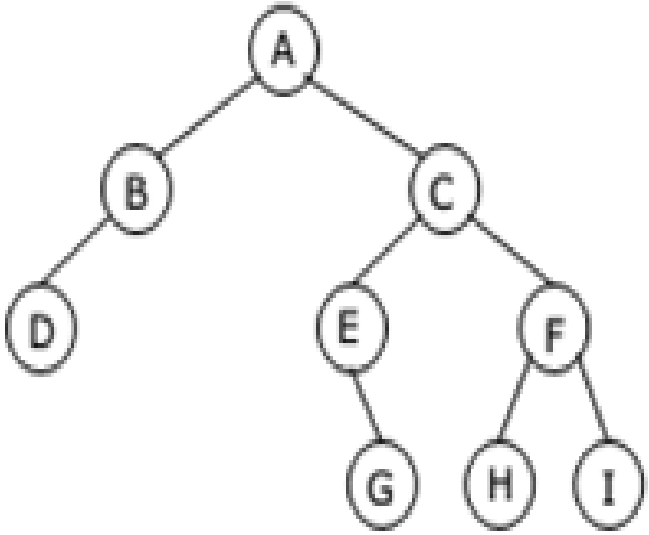
Animation links for BST

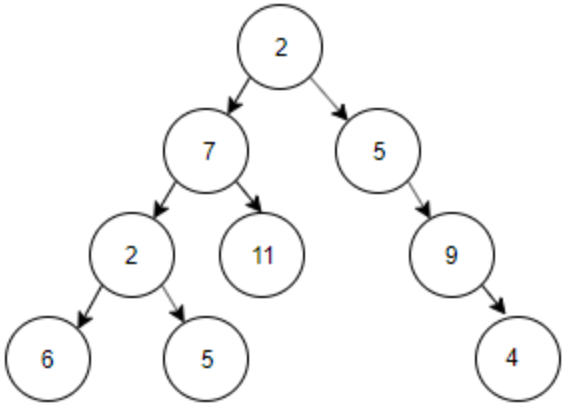
- https://www.cs.usfca.edu/~galles/visualization/BST.html


A
B
C
D
E
F
G


1
2
3
4
5


A
B
C
D
F
G
H
I
J
K


A
B
C
D
E
F
G
H
I


2
7
5
2
11
9
6
5
4

- The number of edges from the node to the deepest leaf is called _____ of the tree.
  a) Height
  b) Depth
  c) Length
  d) Width

- What is a full binary tree?
  a) Each node has exactly zero or two children
  b) Each node has exactly two children
  c) All the leaves are at the same level
  d) Each node has exactly one or two children

- What is a complete binary tree?
  a) Each node has exactly zero or two children
  b) A binary tree, which is completely filled, with the possible exception of the bottom level, which is filled from right to left
  c) A binary tree, which is completely filled, with the possible exception of the bottom level, which is filled from left to right
  d) A tree In which all nodes have degree 2

- Which of the following is not an advantage of trees?
  a) Hierarchical structure
  b) Faster search
  c) Router algorithms
  d) Undo/Redo operations in a notepad

- In a full binary tree if number of internal nodes is I, then number of leaves L are?

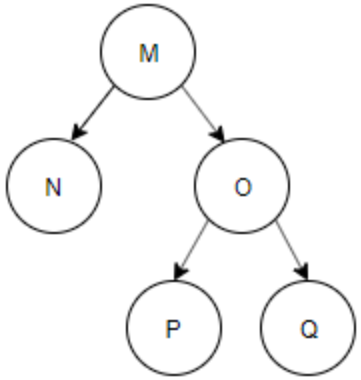
  a) L = 2*I

  b) L = I + 1

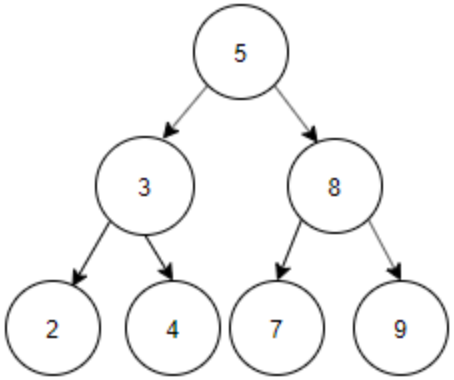
  c) L = I – 1

  d) L = 2*I – 1

- Construct a binary tree by using postorder and inorder sequences given below.
  Inorder: N, M, P, O, Q
  Postorder: N, P, Q, O, M


M
N
O
P
Q

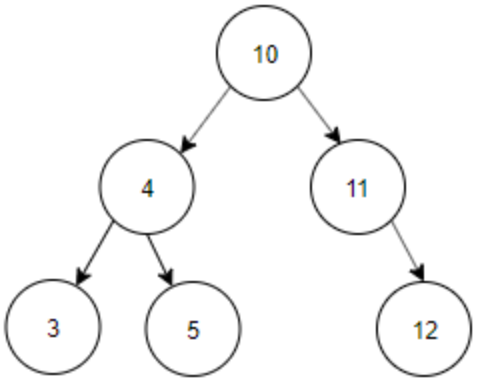
- Construct a binary search tree by using postorder sequence given below.
  Postorder: 2, 4, 3, 7, 9, 8, 5.
- Inorder: 2,3,4,5,7,8,9



- Which of the following is false about a binary search tree?
  a) The left child is always lesser than its parent
  b) The right child is always greater than its parent
  c) The left and right sub-trees should also be binary search trees
  d) In order sequence gives decreasing order of elements

- What is the speciality about the inorder traversal of a binary search tree?
  a) It traverses in a non increasing order
  b) It traverses in an increasing order
  c) It traverses in a random fashion
  d) It traverses based on priority of the node

- Construct a binary search tree with the below information.
  The preorder traversal of a binary search tree 10, 4, 3, 5, 11, 12.
- DFS-preorder
- BFS-level order



$O(1)$	constant
$O(\log n)$	logarithmic
$O(n)$	linear
$O(n \log n)$	"n log n"
$O(n^2)$	quadratic
$O(n^3)$	cubic
$n^{O(1)}$	polynomial
$2^{O(n)}$	exponential

Which of the following sorting algorithm is of divide-and-conquer type?

A. Bubble sort

B. Insertion sort

C. Quick sort

D. All of above

An algorithm that calls itself directly or indirectly is known as

A. Sub algorithm

B. Recursion

C. Polish notation

D. Traversal algorithm

In linked lists there are no NULL links in:

A. Single linked list

B. Linear doubly linked list

C. circular linked list

D. None of the above

The in order traversal of tree will yield a sorted listing of elements of tree in

A. Binary trees

B. Binary search trees

C. Heaps

D. None of above

Recursive algorithms are based on

A. Divideand conquer approach

B. Top-down approach

C. Bottom-up approach

D. Hierarchical approach

E. Heuristic approach.

The result of evaluating prefix expression */b+-dacd, where a = 3, b = 6, c = 1, d = 5 is

A. 0

B. 5

C. 10

D. 15

. The disadvantage in using a circular linked list is .....

A. It is possible to get into infinite loop.

B. Last node points to first node.

C. Time consuming

D. Requires more memory space

What do you call the selected keys in the quick sort method?

A. Outer key

B. Inner Key

C. Partition key

D. Pivot key

E. Recombine key.

4. A linear list in which each node has pointers to point to the predecessor and successors nodes is called as ..

A. Singly Linked List

B. Circular Linked List

C. Doubly Linked List

D. Linear Linked List

- The seven elements A,B,C,D,E,F,G are pushed into stack in a reverse order starting from g. The stack is popped 5 times and each element is inserted into queue. Two elements are deleted from the queue and pushed back onto the stack. Now one element is popped from the stack. The popped item is.....

a) A
b) B
c) F
d) G