Q.1. Explain Tree data structure and various terms related to it.

A tree is defined as a finite set of elements or nodes, such that

1. One of the nodes present at the top of the tree is marked as root node.
2. The remaining elements are partitioned across multiple subtrees present below the root node.

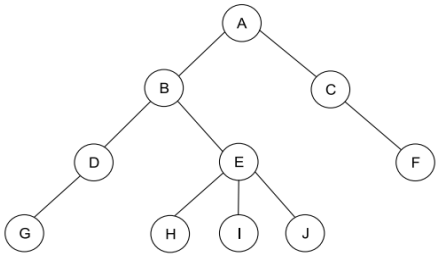


Fig. 8.1 *Tree T*

Here, T is a simple tree containing ten nodes with A being the root node. The node A contains two subtrees. The left subtree starts at node B while the right subtree starts at node C. Both the subtrees further contain subtrees below them, thus indicating recursive nature of the tree data structure. Each node in the tree has zero or more child nodes.

Tree Terminology

Key Term	Description	Example (Refer to Fig. 8.1)
Node	It is the data element of a tree. Apart from storing a value, it also specifies links to the other nodes.	A, B, C, D

Key Term	Description	Example (Refer to Fig. 8.1)
Root	It is the top node in a tree.	A
Parent	A node that has one or more child nodes present below it is referred as parent node.	B is the parent node of D and E
Child	All nodes in a tree except the root node are child nodes of their immediate predecessor nodes.	H, I and J are child nodes of E
Leaf	It is the terminal node that does not have any child nodes.	G, H, I, J and F are leaf nodes
Internal node	All nodes except root and leaf nodes are referred as internal nodes.	B, C, D and E are internal nodes
Sibling	All the child nodes of a parent node are referred as siblings.	D and E are siblings
Degree	The degree of a node is the number of subtrees coming out of the node.	Degree of A is 2 Degree of E is 3
Level	All the tree nodes are present at different levels. Root node is at level 0, its child nodes are at level 1, and so on.	A is at level 0 B and C are at level 1 G, H, I, J are at level 3
Depth or Height	It is the maximum level of a node in the tree.	Depth of tree T is 3
Path	It is the sequence of nodes from source node till destination node.	A–B–E–J

Q.2. Elaborate an Expression Tree.
Ans:

A tree is a kind of data structure that is used to represent the data in hierarchical form. It can be defined as a collection of objects or entities called as nodes that are linked together to simulate a hierarchy. Tree is a non-linear data structure as the data in a tree is not stored linearly or sequentially.

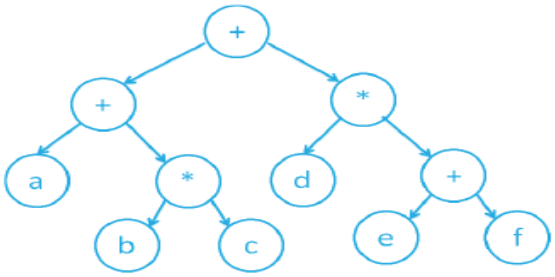
A mathematical expression can be expressed as a binary tree using expression trees. Expression trees are binary trees with each leaf node serving as an operand and each internal (non-leaf) node serving as an operator.

Properties of Expression Tree in Data Structure

- The operands are always represented by the leaf nodes. These operands are always used in the operations.
- The operator at the root of the tree is always given top priority.
- When compared to the operators at the bottom of the tree, the operator at the bottom is always given the lowest priority.
- Because the operand is always present at a depth of the tree, it is given the highest priority of all operators.
- The expression tree can be traversed to evaluate prefix expressions, postfix expressions, and infix expressions.

The given expression can be evaluated using the expression tree in data structure.

```
a + (b * c) + d * (e + f)
```



Uses of an Expression Tree in Data Structure

The following is the primary application of these expression trees in data structure:

- It evaluates, analyses, and modifies diverse phrases. It can also be used to determine the associativity of each operator in an expression.
  As an example:
  The + operator is left-associative, while the / operator is right-associative. The expression trees helped to solve the conundrum of this associativity.
- A context-free grammar is used to create these expression trees.
- It is a key component in compiler design and is part of the semantic analysis step.
- The expression trees are mostly used to create complex expressions that can be quickly evaluated.

Q.3.Explain Binary Search Tree with example

Ans:

A tree is a kind of data structure that is used to represent the data in hierarchical form. It can be defined as a collection of objects or entities called as nodes that are linked together to simulate a hierarchy. Tree is a non-linear data structure as the data in a tree is not stored linearly or sequentially.

A binary tree is referred as a binary search tree if for any node n in the tree:

1. the node elements in the left subtree of n are lesser in value than n.

2. the node elements in the right subtree of n are greater than or equal to n.

Thus, binary search tree arranges its node elements in a sorted manner. As the name suggests, the most important application of a binary search tree is searching. The average running time of searching an element in a binary search tree is O (logn), which is better than other data structures like array and linked lists.
Figure 8.8 shows a sample binary search tree.

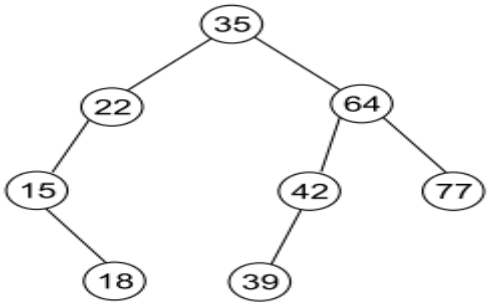


Fig. 8.8 *Binary search tree*

As we can see in the figure, all the nodes in the left subtree are less than the nodes in the right subtree.
The various operations performed on a binary search tree are:
1. Insert
2. Search
3. Delete

1. Insert The insert operation involves adding an element into the binary tree. The location of the new element is determined in such a manner that insertion does not disturb the sort order of the tree.
2. Search The search operation involves traversing the various nodes of the binary tree to search the desired element. For example, if the value to be searched is less than the root value then the remainder of the search operation will only be performed in the left subtree while the right subtree will be completely ignored.
3. Delete The delete operation involves removing an element from the binary search tree.

The delete operation is depicted in Fig. 8.9

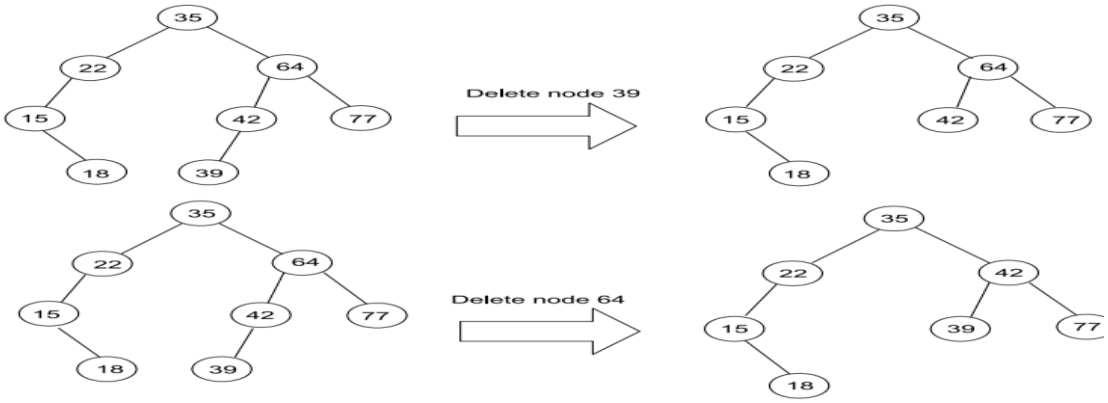


Fig. 8.9 *Deleting an element from binary search tree*

As we can see in Fig. 8.9, if the node to be deleted is a leaf node, then it is simply deleted without requiring any shuffling of other nodes. However, if the node to be deleted is an internal node then appropriate shuffling is required to ensure that the tree regains its sort order.

Example of creating a binary search tree

Now, let's see the creation of binary search tree using an example.

Suppose the data elements are **- 45, 15, 79, 90, 10, 55, 12, 20, 50**

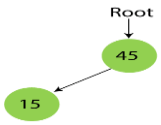
- First, we have to insert **45** into the tree as the root of the tree.
- Then, read the next element; if it is smaller than the root node, insert it as the root of the left subtree, and move to the next element.
- Otherwise, if the element is larger than the root node, then insert it as the root of the right subtree.

Now, let's see the process of creating the Binary search tree using the given data element. The process of creating the BST is shown below -

Step 1 - Insert 45.

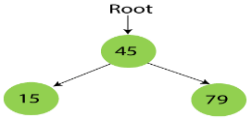
Step 2 - Insert 15.

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



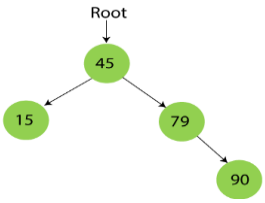
Step 3 - Insert 79.

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



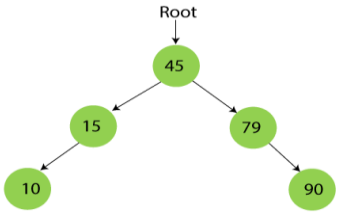
Step 4 - Insert 90.

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



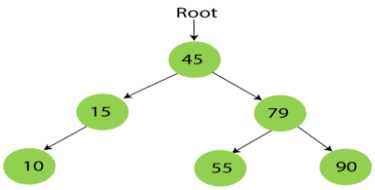
Step 5 - Insert 10.

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



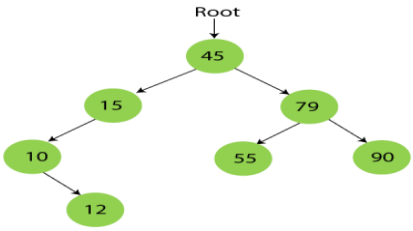
Step 6 - Insert 55.

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



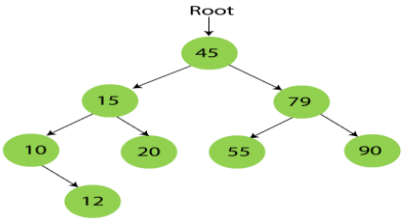
Step 7 - Insert 12.

12 is smaller than 45 and 15 but greater than 10, so it will be inserted as the right subtree of 10.



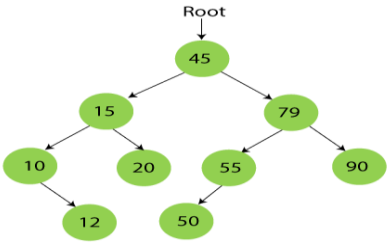
Step 8 - Insert 20.

20 is smaller than 45 but greater than 15, so it will be inserted as the right subtree of 15.



Step 9 - Insert 50.

50 is greater than 45 but smaller than 79 and 55. So, it will be inserted as a left subtree of 55



Q.4. Elaborate m-way tree.
Ans:

A tree is a kind of data structure that is used to represent the data in hierarchical form. It can be defined as a collection of objects or entities called as nodes that are linked together to simulate a hierarchy. Tree is a non-linear data structure as the data in a tree is not stored linearly or sequentially.

Binary search trees are more suitable for smaller data sets where the data is static. However, for large data sets which require dynamic access (example file storage); binary search trees are not exactly suitable. For such cases, the nodes of the tree are required to store large amounts of data. This is achieved with the help of m-way trees.
m-way search trees are an extension of binary search trees having the following properties:

1. Each node of the tree stores 1 to m–1 number of keys.
2. The keys are stored in a sorted manner inside the node.
3. A node containing k values can have a maximum of k+1 subtrees.
4. The subtree pointed by pointer Ti has values less than the key value of ki+1.
5. All the subtrees are m-way trees.

Figure 8.17 shows a sample m-way tree.

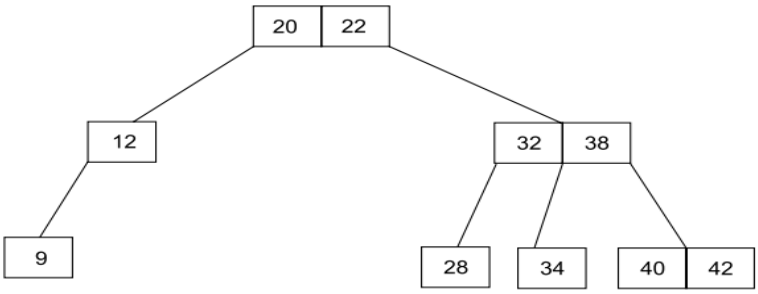


Fig. 8.17 *Sample m-way tree*

(a) B tree To ensure efficiency while searching an m-way tree it is important to control its height. This is achieved with the help of B tree. A B tree is nothing but a height balanced m-way search tree. A B tree of order m has the following properties:
i. Root node is either a leaf node or it contains child nodes ranging from 2 to m.

ii. All internal nodes contain a maximum of m–1 keys.
iii. Number of children of internal nodes ranges from m/2 to m.
iv. Number of keys stored in the leaf nodes ranges from (m–1)/2 to m–1. All the keys are stored in a sorted manner.
v. All leaf nodes are at the same depth.

Figure 8.18 shows a sample B tree.

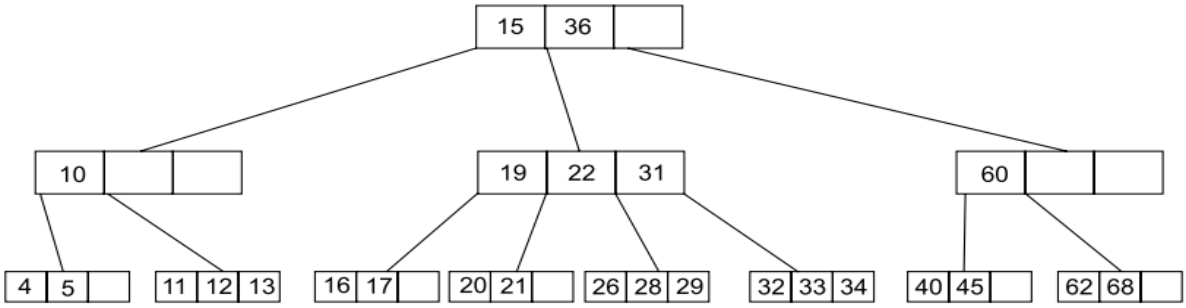


Fig. 8.18 *Sample B tree*

An element is inserted in a B tree by first identifying the location where the new node should be inserted. If the existing node is not full, the new element is inserted within the existing node and an appropriate pointer is created linking it with the parent node. However, if the exiting node is full then it is split into three parts. The middle part is accommodated with the parent node while the new element is inserted in one of the child nodes.
Similarly, deletion of an element from a B tree is done by first removing the element from a node and then carrying out appropriate redistributions to ensure that the tree stays true to its properties.

(b) B+ tree B+ tree is a variant of B tree that is mainly used for implementing index sequential access of records. The main difference between B+ tree and B tree is that in B+ tree data records are only stored in the leaf nodes. The internal nodes of a B+ tree are only used for storing the key values. The key values help in performing the search operation. If the target element is less than a key value then the search proceeds towards its left pointer. Similarly, if the target element is greater than a key value then the search proceeds towards its right pointer.
A B+ tree of order m has the following properties:
i. The internal nodes contain up to m–1 keys.
ii. The number of children of internal nodes lies between m/2 and m.
iii. All leaf nodes are at the same level.
iv. All the leaf nodes are sequentially connected through a linked list.

B+ tree is typically used for implementing index sequential file organization in a database. The internal nodes are used for representing index values through which data records in the sequence set are accessed.
Figure 8.19 shows a sample B+ tree.

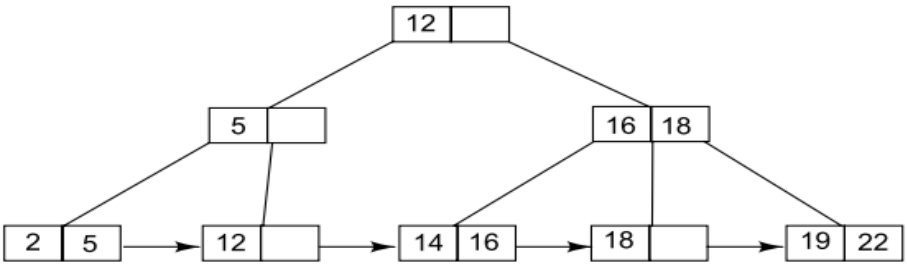


Fig. 8.19 *Sample B⁺ tree*

Q.5. Explain Balanced Trees

Ans:

A tree is a kind of data structure that is used to represent the data in hierarchical form. It can be defined as a collection of objects or entities called as nodes that are linked together to simulate a hierarchy. Tree is a non-linear data structure as the data in a tree is not stored linearly or sequentially.

With each addition of a node in a tree, there is a possibility that the height of the tree may also get changed. The height of a tree has a direct affect on its efficiency to perform the search operation.

For instance, consider the binary search trees shown in Fig. 8.12.

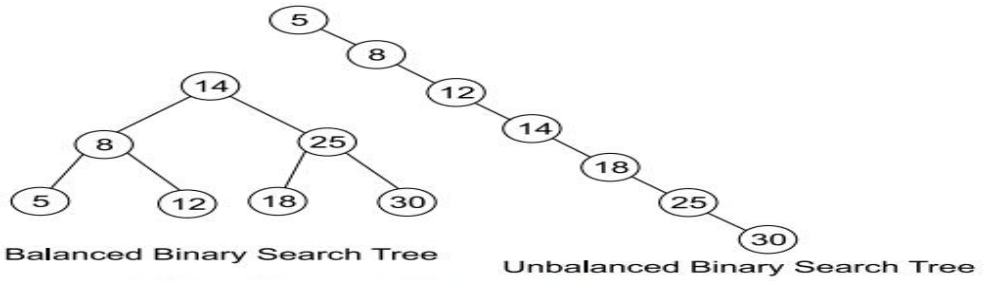


Fig. 8.12 *Binary search trees*

Both the binary search trees shown in the above figure contain the same nodes however the height of the first tree is 2 while that of the second tree is 6. To search element 30 in the above trees, we need to dig a lot deeper in the second tree as compared to the first tree. Thus, while implementing binary trees, it is important to keep the height of the tree in check.

There are various binary search trees that keep the tree balanced whenever a new node is added by shuffling the tree nodes appropriately. These are:

1. AVL tree
2. Red-Black tree

1. AVL tree:AVL tree, also called height-balanced tree was defined by mathematicians Adelson, Velskii and Landis in the year 1962. The main characteristic of an AVL tree is that for all its nodes, the height of the left subtree and the height of the right subtree never differ by more than At any point of time, an AVL tree node is in any one of the following states:

(a) Balanced The height of left subtree is equal to the height of right subtree.
! 4 The height of left subtree is one more than the height of right subtree.
! # The height of right subtree is one more than the height of left subtree.
Figure 8.13 shows an AVL tree.

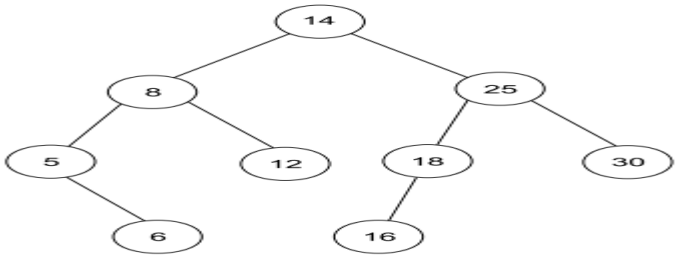


Fig. 8.13 *AVL tree*

As shown in Fig. 8.13, the height of left and right subtrees of each node differs by not more than 1. Now, how is an AVL tree created and maintained? This is done by associating a balance factor (BF) with each node that keeps a track of the height balance for that particular node. BF for a node is calculated by using the following formula:
BF = Height of Left Subtree – Height of Right Subtree
2. Red-Black tree Red-Black tree is a self-balancing binary search tree that has an average running time of O (logn) for insert, delete and search operations. As the name suggests, the red-black tree associates a color attribute with each node, which can possess only two values, red or black. That means each node in a red-black tree is either red or black colored. Apart from possessing the properties of a typical binary search tree, a red-black tree possesses the following properties:
(a) Each node is either red or black in color.
(b) The root node is black colored.
(c) The leaf nodes are black colored. It includes the NULL children.
(d) The child nodes of all red-colored nodes are black.
(e) Each path from a given node to any of its leaf nodes contains equal number of black nodes. The number of such black nodes is also referred as black-height (bh) of the node.
The above properties ensure that the length of the longest path from the root node to a leaf node is less than roughly twice of the shortest path. This ensures that the balance of the tree is always kept under check. The key advantage of a red-black tree is that its worst case running time is better than most of the other binary search trees.
Figure 8.16 shows a red-black tree.

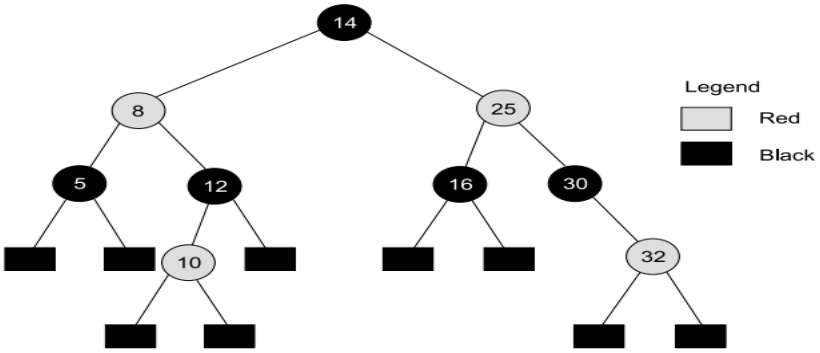


Fig. 8.16 *Red-Black tree*

The insert and delete operations on a red-black tree require small number of rotations as well as change of colors of some of the nodes so that the tree complies with all the properties of a red-black tree. However, the average running time of these operations is O (logn)

Q.6. Explain Binary tree traversal.
Ans:

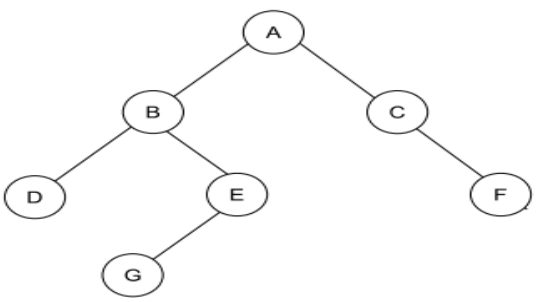
Traversal is the process of visiting the various elements of a data structure. Binary tree traversal can be performed using three methods:
1. Preorder
2. Inorder
3. Postorder

1. Preorder The preorder traversal method performs the following operations:
(a) Process the root node (N).
(b) Traverse the left subtree of N (L).
(c) Traverse the right subtree of N (R).

2. Inorder The inorder traversal method performs the following operations:
(a) Traverse the left subtree of N (L).
(b) Process the root node (N).
(c) Traverse the right subtree of N (R).

3. Postorder The postorder traversal method performs the following operations:(a) Traverse the left subtree of N (L).
(b) Traverse the right subtree of N (R).
(c) Process the root node (N).

Figure 8.7 shows an illustration of the different binary tree traversal methods.
Consider the following binary tree:



Solution

(a) Preorder traversal sequence
A–B–D–E–G–C–F

(b) Inorder traversal sequence
D–B–G–E–A–C–F

(c) Postorder traversal sequence
D–G–E–B–F–C–A