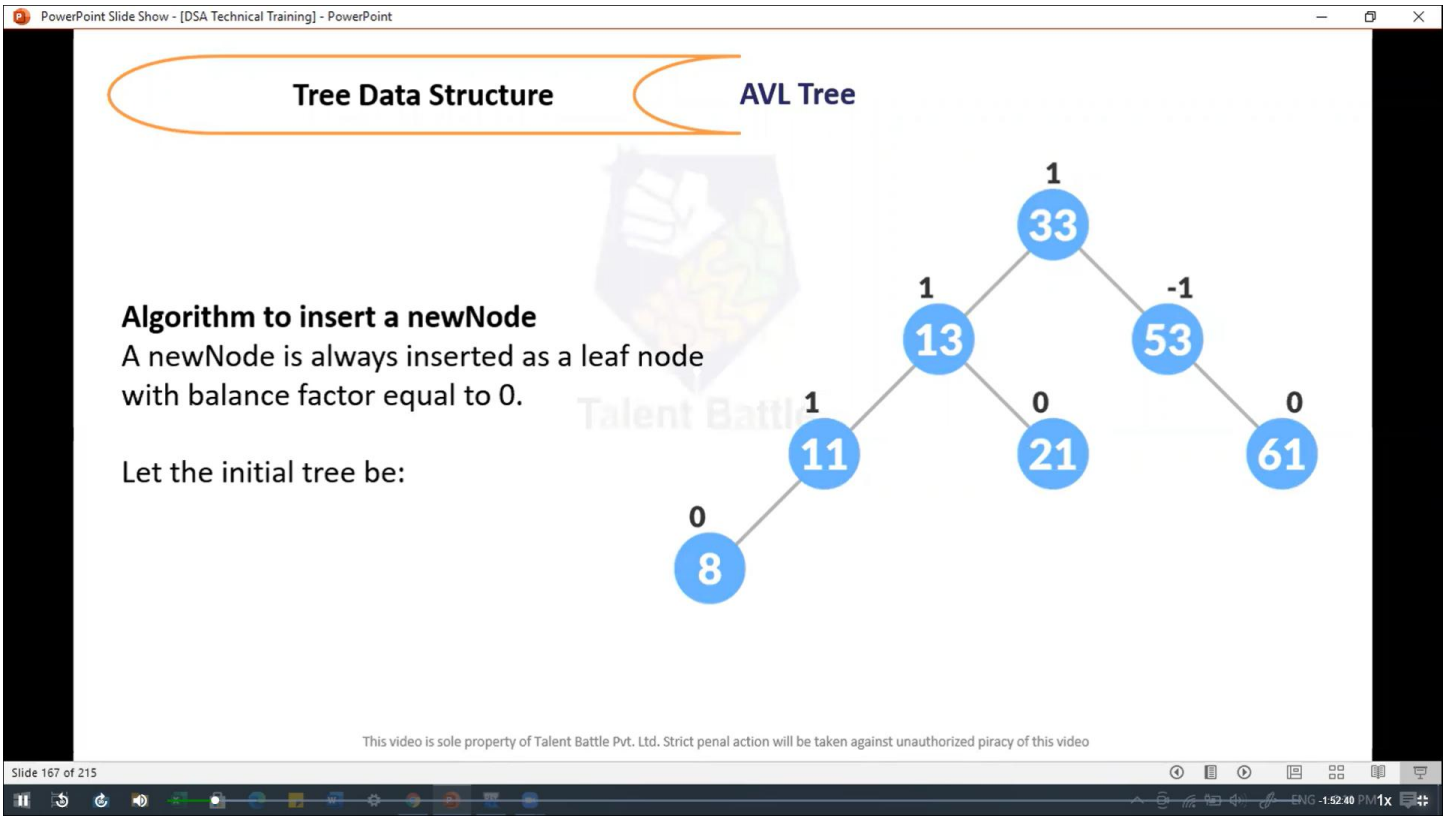
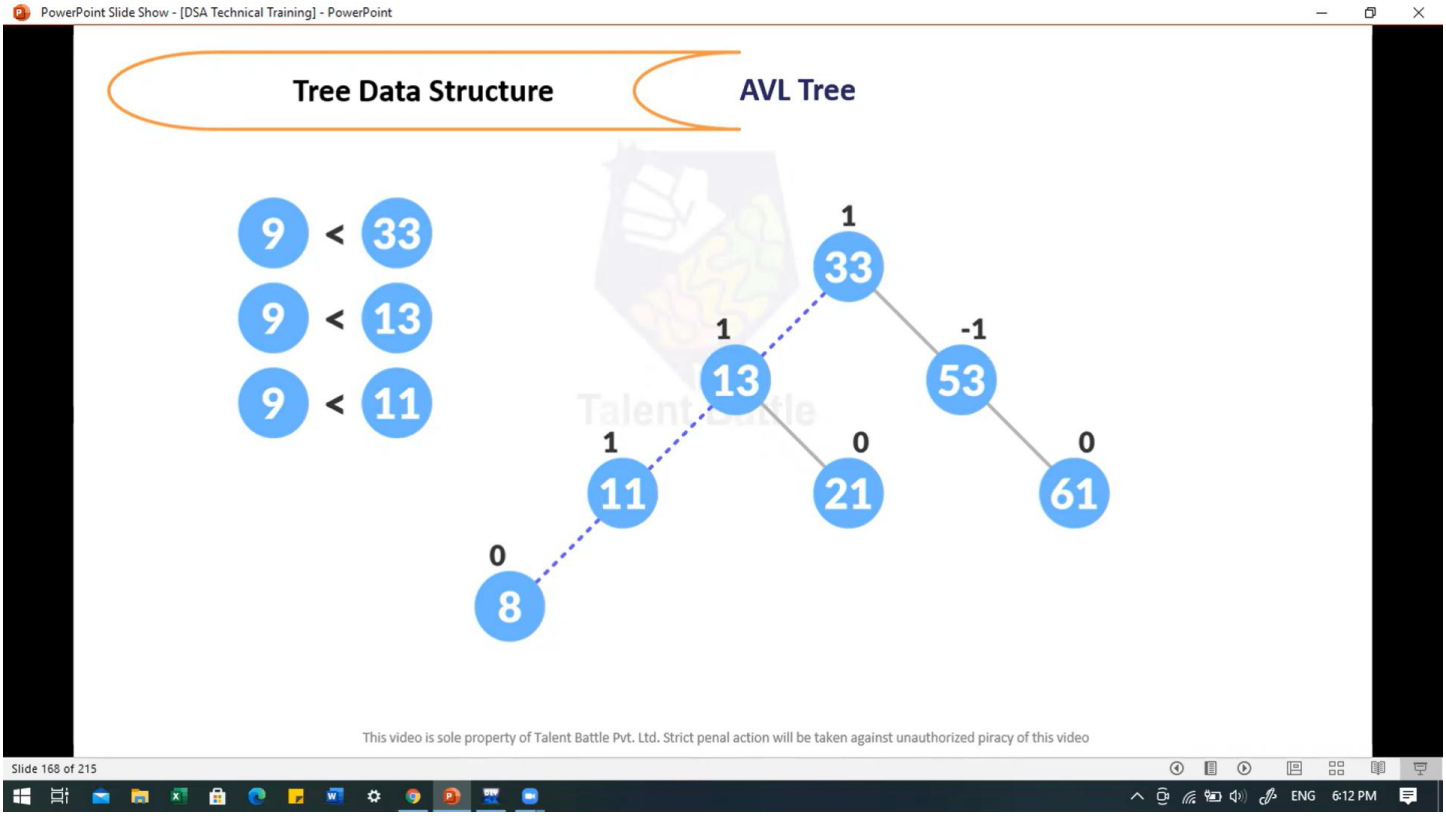
Session9

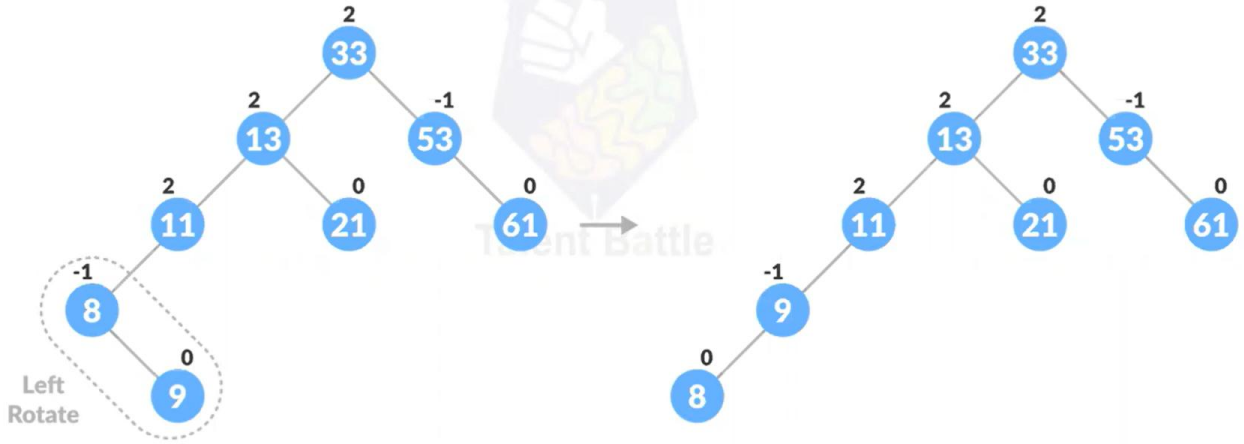
Tree data structure3





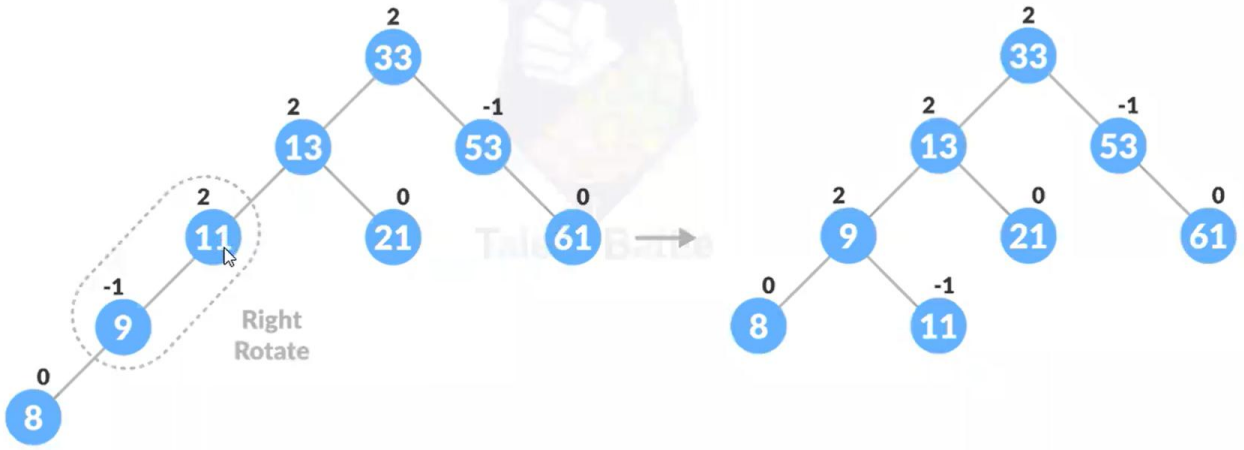
PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure — AVL Tree



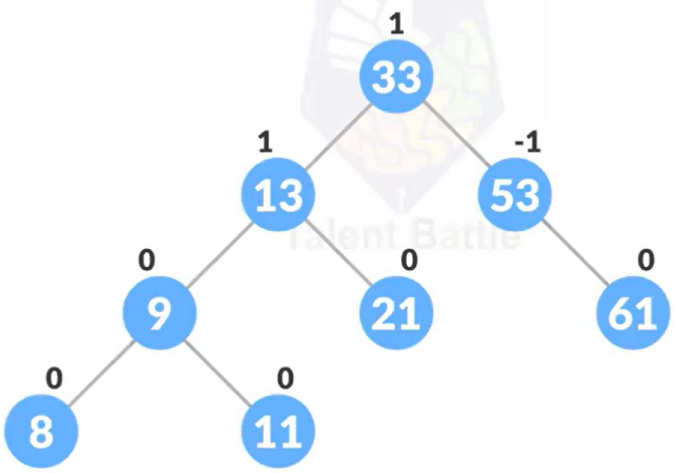
PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure — AVL Tree



PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure — AVL Tree



Final balanced tree

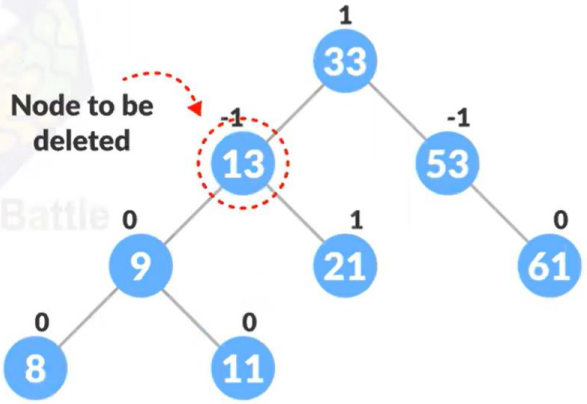
ENG 6:17 PM

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure — AVL Tree

Algorithm to Delete a node

A node is always deleted as a leaf node.
After deleting a node, the balance factors of the nodes get changed. In order to rebalance the balance factor, suitable rotations are performed.



ENG 6:20 PM

Tree Data Structure — AVL Tree

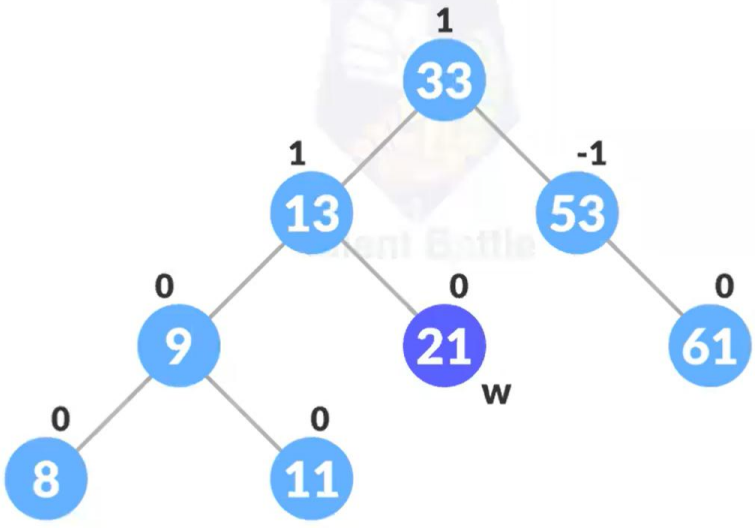
There are three cases for deleting a node:

1. If nodeToBeDeleted is the leaf node (ie. does not have any child), then remove nodeToBeDeleted.

2. If nodeToBeDeleted has one child, then substitute the contents of nodeToBeDeleted with that of the child. Remove the child.

3. If nodeToBeDeleted has two children, find the inorder successor w of nodeToBeDeleted (ie. node with a minimum value of key in the right subtree).

Tree Data Structure — AVL Tree

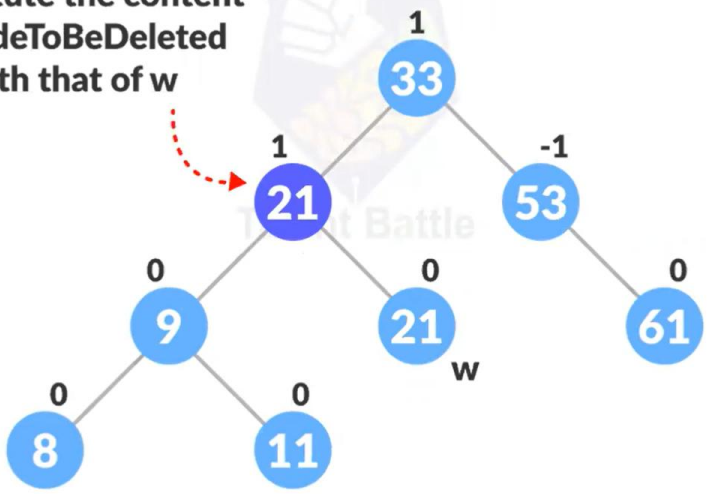


PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure

AVL Tree

Substitute the content of nodeToBeDeleted with that of w

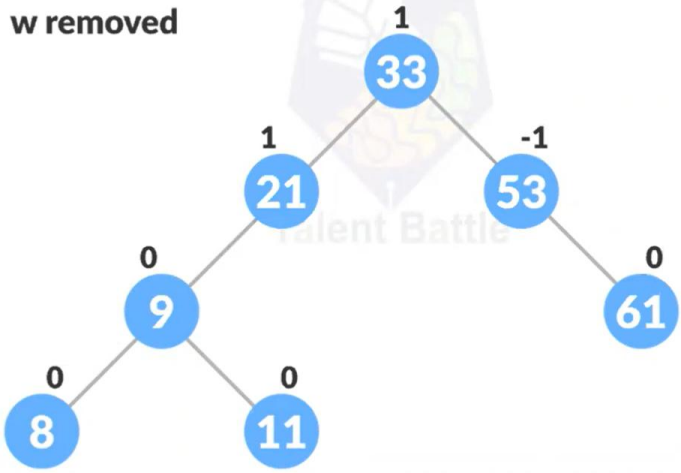


PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure

AVL Tree

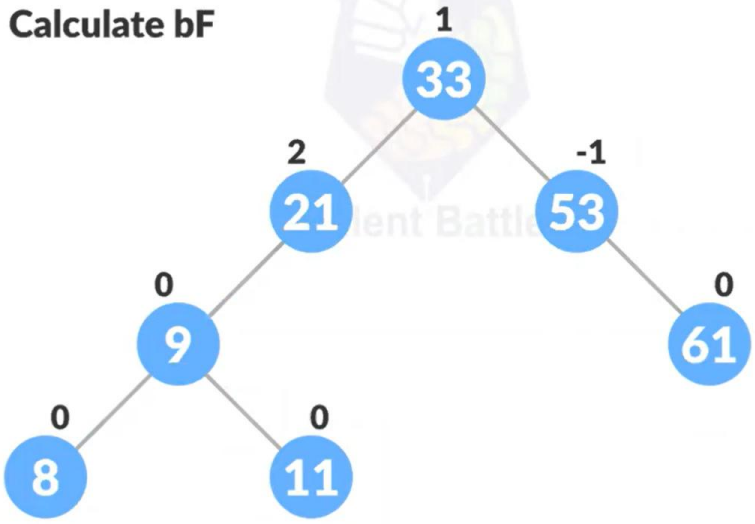
w removed



PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure | AVL Tree

Calculate bF

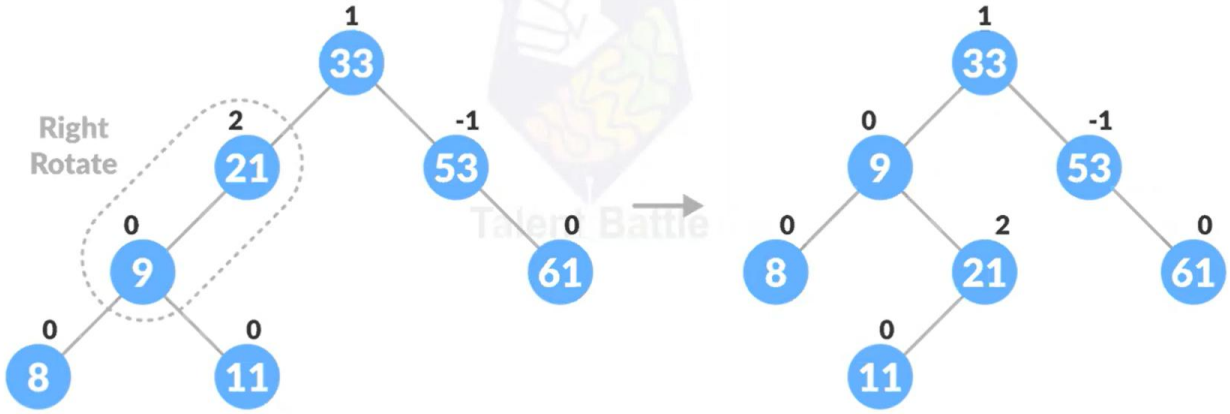


Slide 179 of 215

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure | AVL Tree

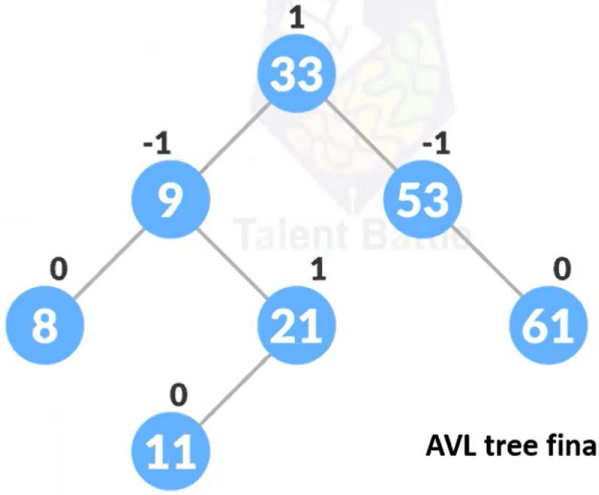
Right Rotate



Slide 180 of 215

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure — AVL Tree



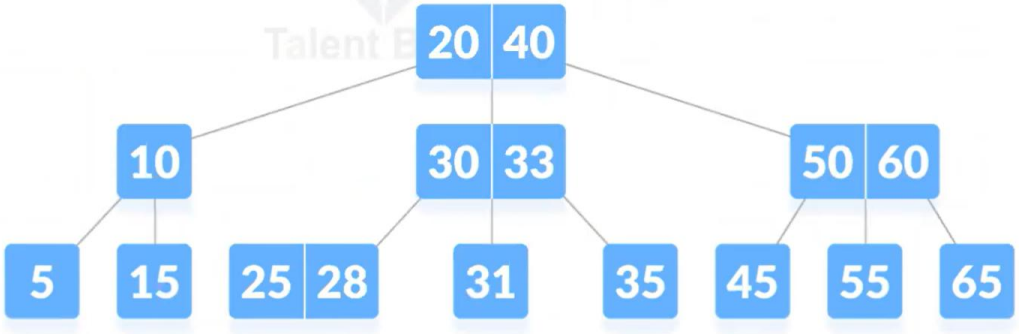
AVL tree final

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure — B - Tree

B-tree is a special type of self-balancing search tree in which each node can contain more than one key and can have more than two children. It is a generalized form of the binary search tree.

It is also known as a height-balanced m-way tree.



Tree Data Structure | B - Tree

Why B-tree?

The need for B-tree arose with the rise in the need for lesser time in accessing the physical storage media like a hard disk. The secondary storage devices are slower with a larger capacity. There was a need for such types of data structures that minimize the disk accesses. Other data structures such as a binary search tree, avl tree, red-black tree, etc can store only one key in one node. If you have to store a large number of keys, then the height of such trees becomes very large and the access time increases.

However, B-tree can store many keys in a single node and can have multiple child nodes. This decreases the height significantly allowing faster disk accesses.

Tree Data Structure | B - Tree

B-tree Properties

1.For each node x, the keys are stored in increasing order.

2.In each node, there is a boolean value x.leaf which is true if x is a leaf.

3.If n is the order of the tree, each internal node can contain at most n - 1 keys along with a pointer to each child.

4.Each node except root can have at most n children and at least n/2 children.

5.All leaves have the same depth (i.e. height-h of the tree).

6.The root has at least 2 children and contains a minimum of 1 key.

Tree Data Structure | **B - Tree**

Operations

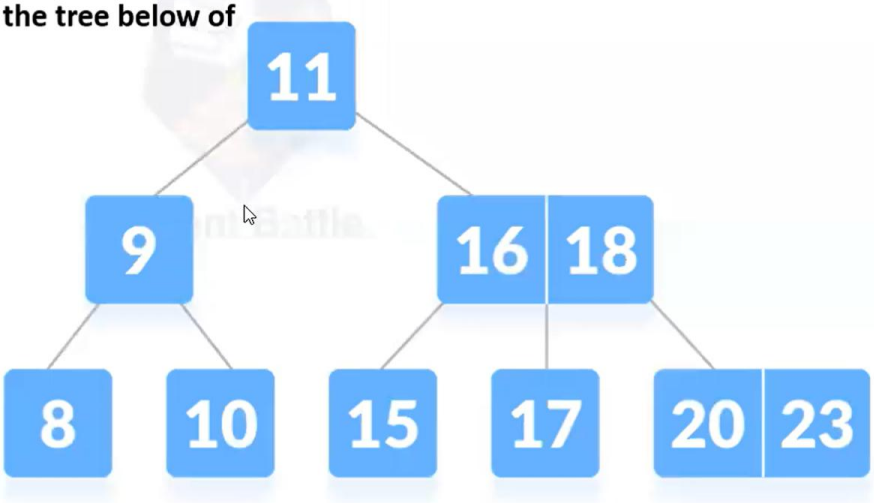
Searching

Searching for an element in a B-tree is the generalized form of searching an element in a Binary Search Tree. The following steps are followed.

1. Starting from the root node, compare k with the first key of the node.
2. If k = the first key of the node, return the node and the index.
3. If k.leaf = true, return NULL (i.e. not found).
4. If k < the first key of the root node, search the left child of this key recursively.
5. If there is more than one key in the current node and k > the first key, compare k with the next key in the node.
6. If k < next key, search the left child of this key (ie. k lies in between the first and the second keys).
   Else, search the right child of the key.
7. Repeat steps 1 to 4 until the leaf is reached.

Tree Data Structure | **B - Tree**

Let us search key k = 17 in the tree below of degree 3

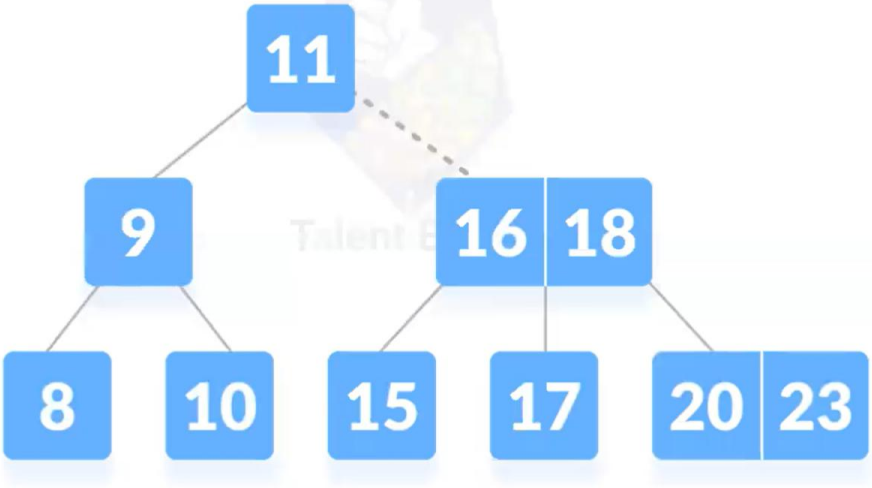


PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure — B - Tree

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

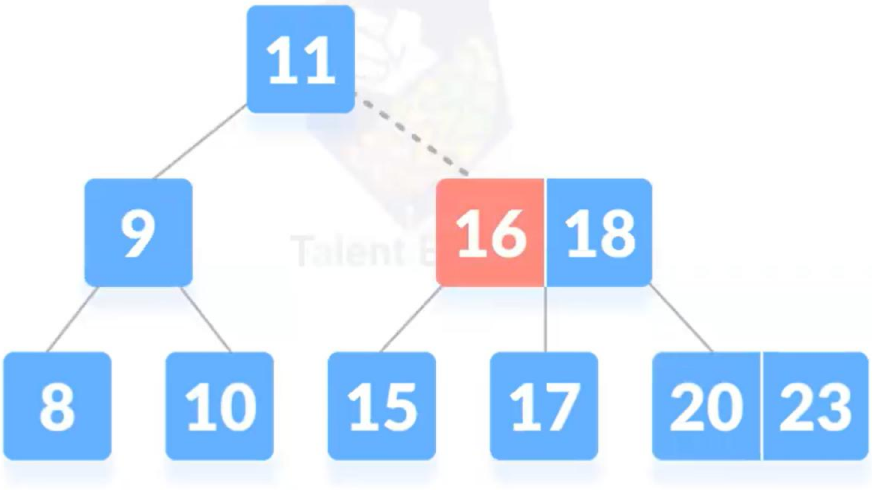
Tree Data Structure — B - Tree



PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure

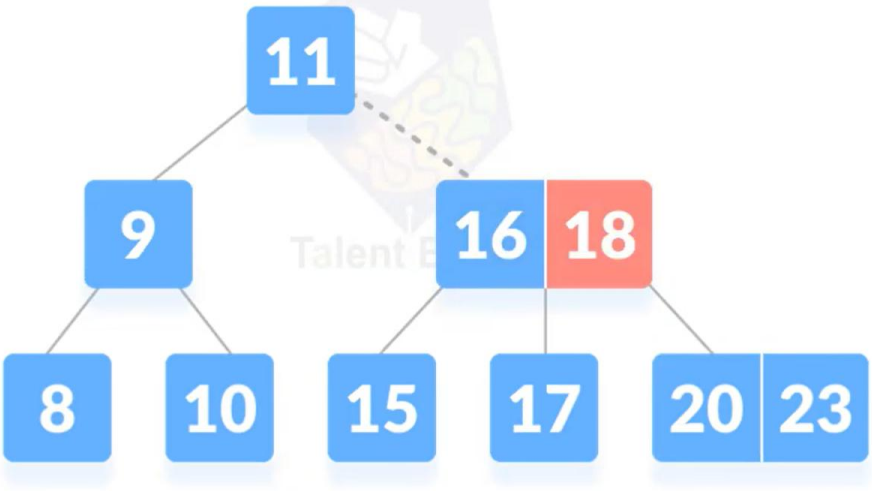
B - Tree



PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure

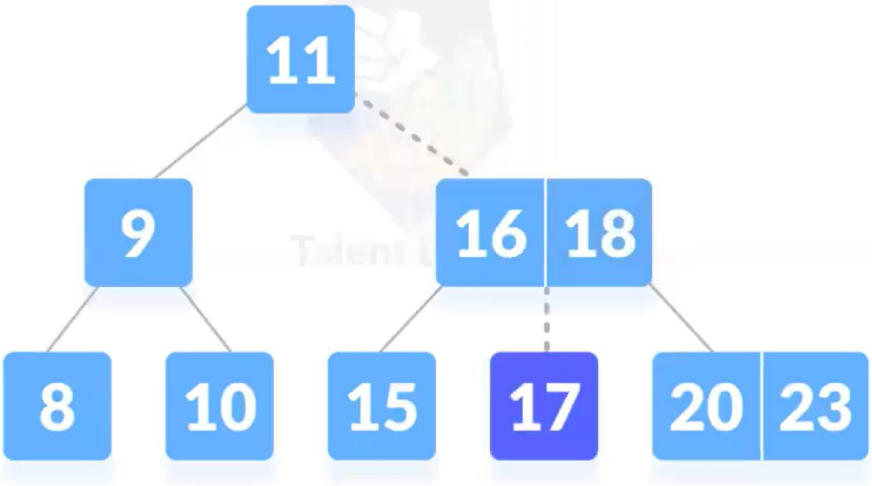
B - Tree



This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure — B - Tree



PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure — B - Tree

Searching Complexity on B Tree

Worst case Time complexity: Θ(log n)

Average case Time complexity: Θ(log n)

Best case Time complexity: Θ(log n)

Average case Space complexity: Θ(n)

Worst case Space complexity: Θ(n)

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure — B - Tree

B Tree Applications

- databases and file systems
- to store blocks of data (secondary storage media)
- multilevel indexing

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure — B + Tree

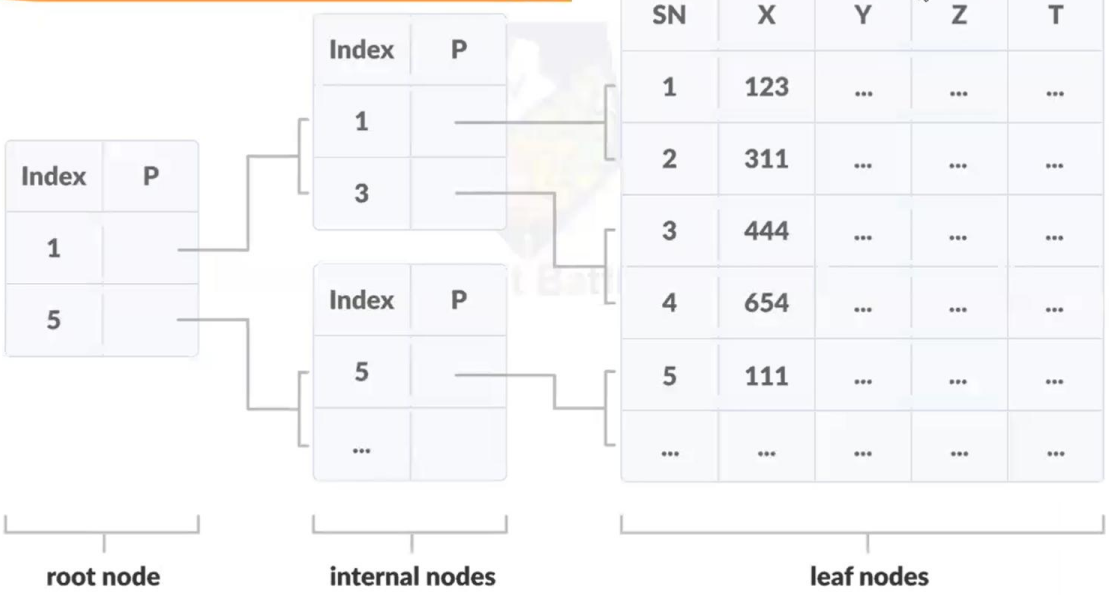
A B+ tree is an advanced form of a self-balancing tree in which all the values are present in the leaf level.

An important concept to be understood before learning B+ tree is multilevel indexing.

It makes accessing the data easier and faster.

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure — B + Tree



PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure — B + Tree

Properties of a B+ Tree

1.All leaves are at the same level.

2.The root has at least two children.

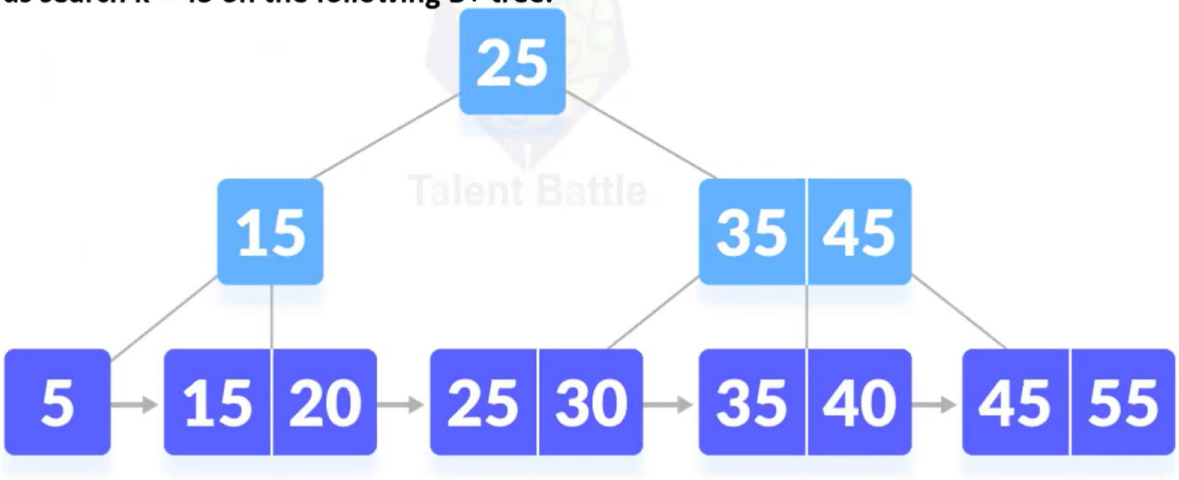
3.Each node except root can have a maximum of m children and at least m/2 children.

4.Each node can contain a maximum of m - 1 keys and a minimum of [m/2] - 1 keys.

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

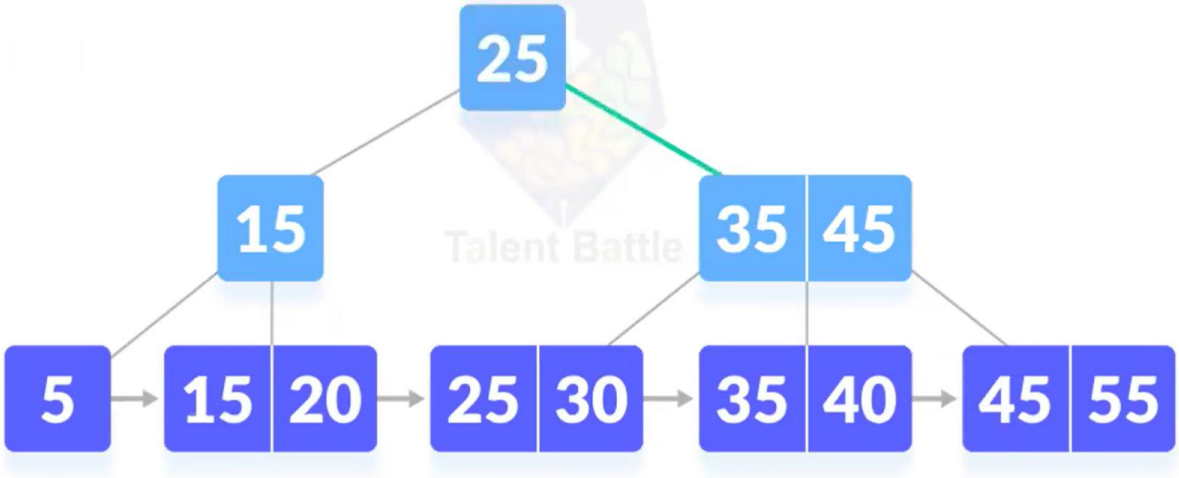
Tree Data Structure — B + Tree

Searching Example on a B+ Tree
Let us search k = 45 on the following B+ tree.



PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure — B + Tree



Tree Data Structure | B + Tree

B+ Tree Applications

- Multilevel Indexing
- Faster operations on the tree (insertion, deletion, search)
- Database indexing

Tree Data Structure | B + Tree

Insertion on a B+ Tree

Inserting an element into a **B+ tree** consists of three main events:

searching the appropriate leaf,

inserting the element and

balancing/splitting the tree.

Tree Data Structure — B + Tree

Insertion Operation

Before inserting an element into a B+ tree, these properties must be kept in mind.

- The root has at least two children.
- Each node except root can have a maximum of m children and at least m/2 children.
- Each node can contain a maximum of m - 1 keys and a minimum of ⌈m/2⌉ - 1 keys.

The following steps are followed for inserting an element.

1. Since every element is inserted into the leaf node, go to the appropriate leaf node.
2. Insert the key into the leaf node.

Case I

1. If the leaf is not full, insert the key into the leaf node in increasing order.

Case II

1. If the leaf is full, insert the key into the leaf node in increasing order and balance the tree in the following way.
2. Break the node at m/2th position.
3. Add m/2th key to the parent node as well.
4. If the parent node is already full, follow steps 2 to 3.

Tree Data Structure — B + Tree

The elements to be inserted are 5,15, 25, 35, 45.

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

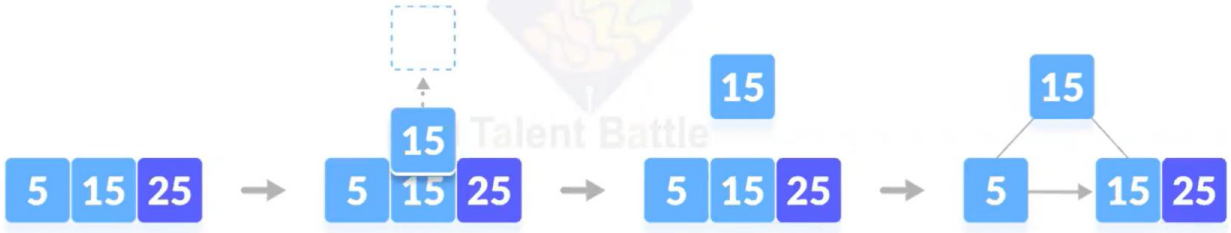
Tree Data Structure

B + Tree

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure

B + Tree



PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

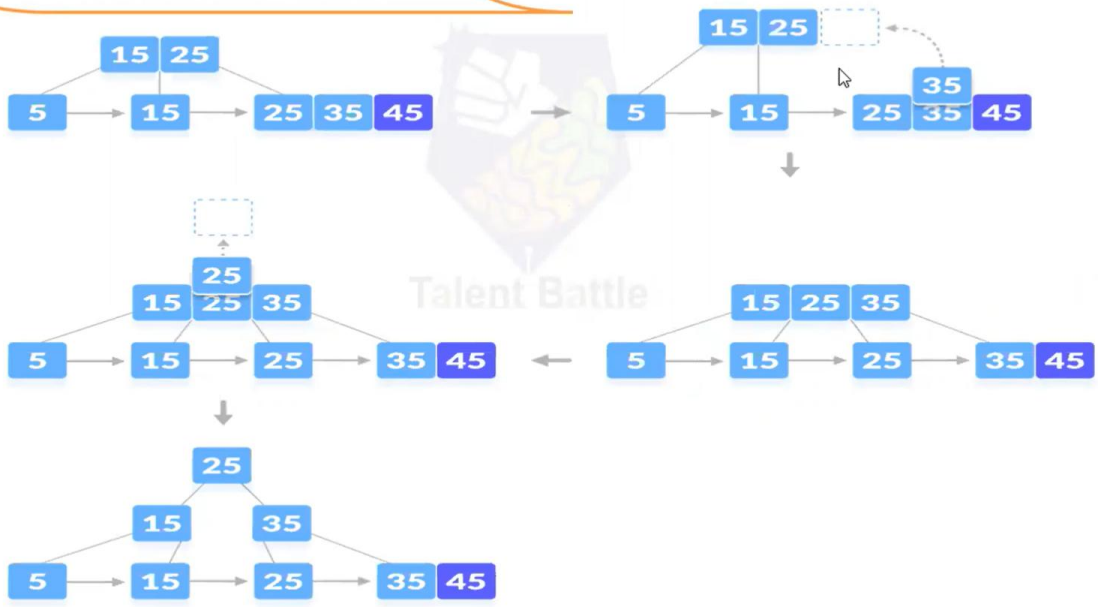
Tree Data Structure

B + Tree

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure

B + Tree



PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure — Red Black Tree

Red-Black tree is a self-balancing binary search tree in which each node contains an extra bit for denoting the color of the node, either red or black.

A red-black tree satisfies the following properties:

1. **Red/Black Property:** Every node is colored, either red or black.
2. **Root Property:** The root is black.
3. **Leaf Property:** Every leaf (NIL) is black.
4. **Red Property:** If a red node has children then, the children are always black.
5. **Depth Property:** For each node, any simple path from this node to any of its descendant leaf has the same black-depth (the number of black nodes).

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

Tree Data Structure — Red Black Tree

Each node has the following attributes:

- color
- key
- leftChild
- rightChild
- parent (except root node)

Tree Data Structure — Red Black Tree

How the red-black tree maintains the property of self-balancing?

The red-black color is meant for balancing the tree.

The limitations put on the node colors ensure that any simple path from the root to a leaf is not more than twice as long as any other such path. It helps in maintaining the self-balancing property of the red-black tree.

Tree Data Structure — Red Black Tree

Red-Black Tree Applications

1. To implement finite maps
2. To implement Java packages: java.util.TreeMap and java.util.TreeSet
3. To implement Standard Template Libraries (STL) in C++: multiset, map, multimap in Linux Kernel

Tree Data Structure — Red Black Tree

Red-Black Tree Applications

1. To implement finite maps
2. To implement Java packages: java.util.TreeMap and java.util.TreeSet
3. To implement Standard Template Libraries (STL) in C++: multiset, map, multimap in Linux Kernel

Tree Data Structure

practice Examples:

1. **WAP to find repeating elements in an array.**

2. Find the 15th term of the series?
0,0,7,6,14,12,21,18, 28

Explanation :
In this series the odd term is increment of 7 {0, 7, 14, 21, 28, 35 – – – – – – }
And even term is a increment of 6 {0, 6, 12, 18, 24, 30 – – – – – – }