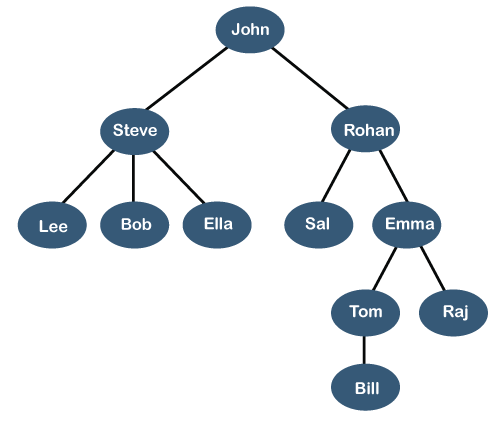
**TREES**

***A tree* is also one of the data structures that represent hierarchical data. Suppose we want to show the employees and their positions in the hierarchical form then it can be represented as shown below:**

****

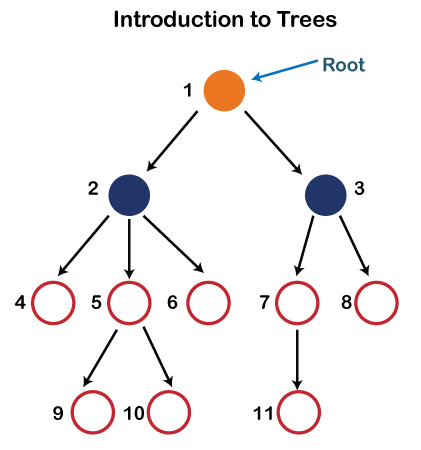
**The above tree shows the organization hierarchy of some company. In the above structure, *john* is the CEO of the company, and John has two direct reports named as *Steve* and *Rohan*. Steve has three direct reports named *Lee, Bob, Ella* where *Steve* is a manager. Bob has two direct reports named *Sal* and *Emma*. Emma has two direct reports named *Tom* and *Raj*. Tom has one direct report named *Bill*. This particular logical structure is known as a *Tree*. Its structure is similar to the real tree, so it is named a *Tree*. In this structure, the *root* is at the top, and its branches are moving in a downward direction. Therefore, we can say that the Tree data structure is an efficient way of storing the data in a hierarchical way.SQL CREATE TABLE**

**Let's understand some key points of the Tree data structure.**

* **A tree data structure is defined as a collection of objects or entities known as nodes that are linked together to represent or simulate hierarchy.**
* **A tree data structure is a non-linear data structure because it does not store in a sequential manner. It is a hierarchical structure as elements in a Tree are arranged in multiple levels.**
* **In the Tree data structure, the topmost node is known as a root node. Each node contains some data, and data can be of any type. In the above tree structure, the node contains the name of the employee, so the type of data would be a string.**
* **Each node contains some data and the link or reference of other nodes that can be called children.**

**Some basic terms used in Tree data structure.**

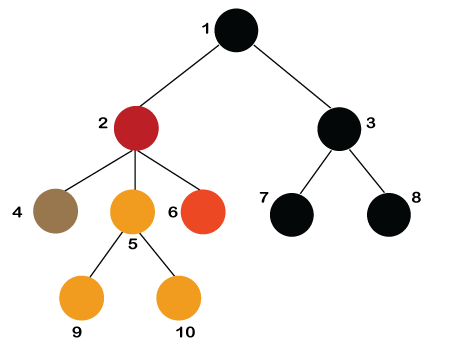
**Let's consider the tree structure, which is shown below:**

****

**In the above structure, each node is labeled with some number. Each arrow shown in the above figure is known as a *link* between the two nodes.**

* **Root: The root node is the topmost node in the tree hierarchy. In other words, the root node is the one that doesn't have any parent. In the above structure, node numbered 1 is the root node of the tree. If a node is directly linked to some other node, it would be called a parent-child relationship.**
* **Child node: If the node is a descendant of any node, then the node is known as a child node.**
* **Parent: If the node contains any sub-node, then that node is said to be the parent of that sub-node.**
* **Sibling: The nodes that have the same parent are known as siblings.**
* **Leaf Node:- The node of the tree, which doesn't have any child node, is called a leaf node. A leaf node is the bottom-most node of the tree. There can be any number of leaf nodes present in a general tree. Leaf nodes can also be called external nodes.**
* **Internal nodes: A node has atleast one child node known as an *internal***
* **Ancestor node:- An ancestor of a node is any predecessor node on a path from the root to that node. The root node doesn't have any ancestors. In the tree shown in the above image, nodes 1, 2, and 5 are the ancestors of node 10.**
* **Descendant: The immediate successor of the given node is known as a descendant of a node. In the above figure, 10 is the descendant of node 5.**

**Properties of Tree data structure**

* **Recursive data structure: The tree is also known as a *recursive data structure*. A tree can be defined as recursively because the distinguished node in a tree data structure is known as a *root node*. The root node of the tree contains a link to all the roots of its subtrees. The left subtree is shown in the yellow color in the below figure, and the right subtree is shown in the red color. The left subtree can be further split into subtrees shown in three different colors. Recursion means reducing something in a self-similar manner. So, this recursive property of the tree data structure is implemented in various applications.  
   **
* **Number of edges: If there are n nodes, then there would n-1 edges. Each arrow in the structure represents the link or path. Each node, except the root node, will have atleast one incoming link known as an edge. There would be one link for the parent-child relationship.**
* **Depth of node x: The depth of node x can be defined as the length of the path from the root to the node x. One edge contributes one-unit length in the path. So, the depth of node x can also be defined as the number of edges between the root node and the node x. The root node has 0 depth.**
* **Height of node x: The height of node x can be defined as the longest path from the node x to the leaf node.**

**Based on the properties of the Tree data structure, trees are classified into various categories.**

**Implementation of Tree**

**The tree data structure can be created by creating the nodes dynamically with the help of the pointers. The tree in the memory can be represented as shown below:**

****

**The above figure shows the representation of the tree data structure in the memory. In the above structure, the node contains three fields. The second field stores the data; the first field stores the address of the left child, and the third field stores the address of the right child.**

**In programming, the structure of a node can be defined as:**

1. **struct node**
2. **{**
3. **int data;**
4. **struct node \*left;**
5. **struct node \*right;**
6. **}**

**The above structure can only be defined for the binary trees because the binary tree can have utmost two children, and generic trees can have more than two children. The structure of the node for generic trees would be different as compared to the binary tree.**

**Applications of trees**

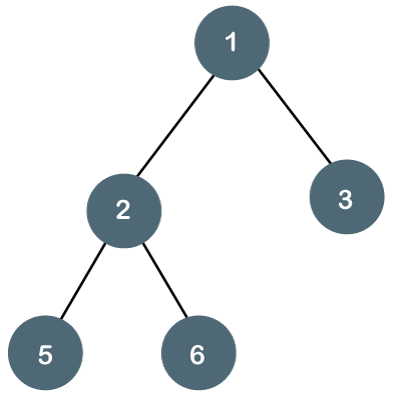
**The following are the applications of trees:**

* **Storing naturally hierarchical data: Trees are used to store the data in the hierarchical structure. For example, the file system. The file system stored on the disc drive, the file and folder are in the form of the naturally hierarchical data and stored in the form of trees.**
* **Organize data: It is used to organize data for efficient insertion, deletion and searching. For example, a binary tree has a logN time for searching an element.**
* **Trie: It is a special kind of tree that is used to store the dictionary. It is a fast and efficient way for dynamic spell checking.**
* **Heap: It is also a tree data structure implemented using arrays. It is used to implement priority queues.**
* **B-Tree and B+Tree: B-Tree and B+Tree are the tree data structures used to implement indexing in databases.**
* **Routing table: The tree data structure is also used to store the data in routing tables in the routers.**

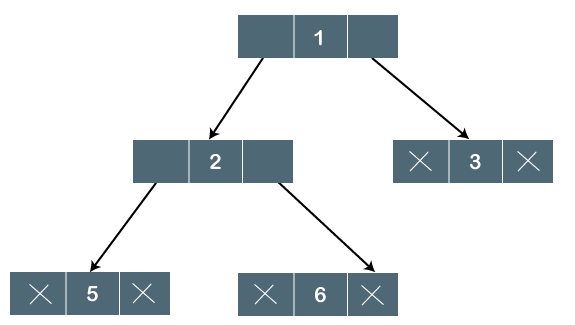
# **Binary Tree**

**The Binary tree means that the node can have maximum two children. Here, binary name itself suggests that 'two'; therefore, each node can have either 0, 1 or 2 children.**

**Let's understand the binary tree through an example.**

****

**The above tree is a binary tree because each node contains the utmost two children. The logical representation of the above tree is given below:**

****

**In the above tree, node 1 contains two pointers, i.e., left and a right pointer pointing to the left and right node respectively. The node 2 contains both the nodes (left and right node); therefore, it has two pointers (left and right). The nodes 3, 5 and 6 are the leaf nodes, so all these nodes contain NULL pointer on both left and right parts.**

### Properties of Binary Tree

* **At each level of i, the maximum number of nodes is 2i.**
* **The height of the tree is defined as the longest path from the root node to the leaf node. The tree which is shown above has a height equal to 3. Therefore, the maximum number of nodes at height 3 is equal to (1+2+4+8) = 15. In general, the maximum number of nodes possible at height h is (20 + 21 + 22+….2h) = 2h+1 -1.**
* **The minimum number of nodes possible at height h is equal to h+1.**
* **If the number of nodes is minimum, then the height of the tree would be maximum. Conversely, if the number of nodes is maximum, then the height of the tree would be minimum.**

**If there are 'n' number of nodes in the binary tree.**

**The minimum height can be computed as:**

**As we know that,**

**n = 2h+1 -1**

**n+1 = 2h+1**

**Taking log on both the sides,**

**log2(n+1) = log2(2h+1)**

**log2(n+1) = h+1**

**h = log2(n+1) - 1**

**The maximum height can be computed as:**

**As we know that,**

**n = h+1**

**h= n-1**

### What is a Binary Search tree?

**A binary search tree follows some order to arrange the elements. In a Binary search tree, the value of left node must be smaller than the parent node, and the value of right node must be greater than the parent node. This rule is applied recursively to the left and right subtrees of the root.**

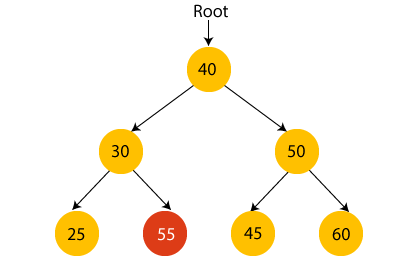
**Let's understand the concept of Binary search tree with an example.**

****

**In the above figure, we can observe that the root node is 40, and all the nodes of the left subtree are smaller than the root node, and all the nodes of the right subtree are greater than the root node.**

**Similarly, we can see the left child of root node is greater than its left child and smaller than its right child. So, it also satisfies the property of binary search tree. Therefore, we can say that the tree in the above image is a binary search tree.**

**Suppose if we change the value of node 35 to 55 in the above tree, check whether the tree will be binary search tree or not.**

****

**In the above tree, the value of root node is 40, which is greater than its left child 30 but smaller than right child of 30, i.e., 55. So, the above tree does not satisfy the property of Binary search tree. Therefore, the above tree is not a binary search tree.**

### Advantages of Binary search tree

* **Searching an element in the Binary search tree is easy as we always have a hint that which subtree has the desired element.**
* **As compared to array and linked lists, insertion and deletion operations are faster in BST.**

### 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.**

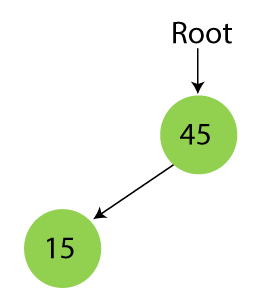
**Now, let's see the process of creating the Binary search tree using the given data element. The process of creating the BST is shown below -**

**Step 1 - Insert 45.**

**Binary Search tree**

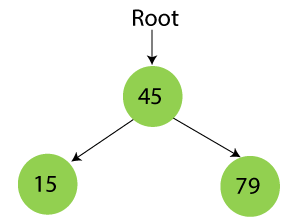
**Step 2 - Insert 15.**

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

****

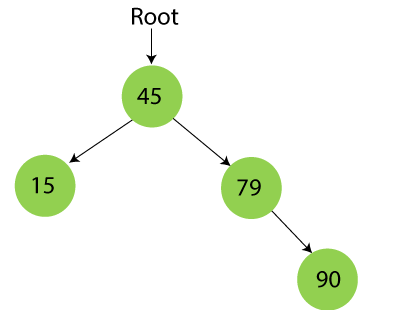
**Step 3 - Insert 79.**

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

****

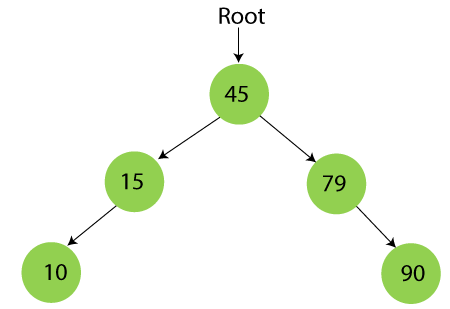
**Step 4 - Insert 90.**

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

****

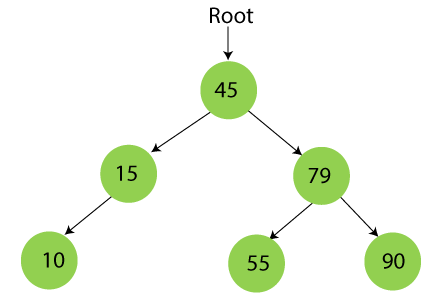
**Step 5 - Insert 10.**

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

****

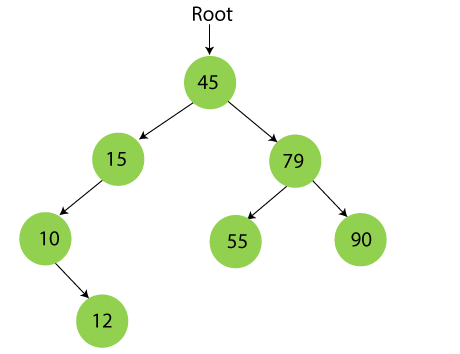
**Step 6 - Insert 55.**

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

****

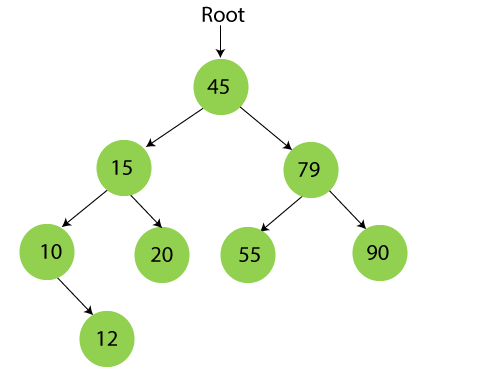
**Step 7 - Insert 12.**

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

****

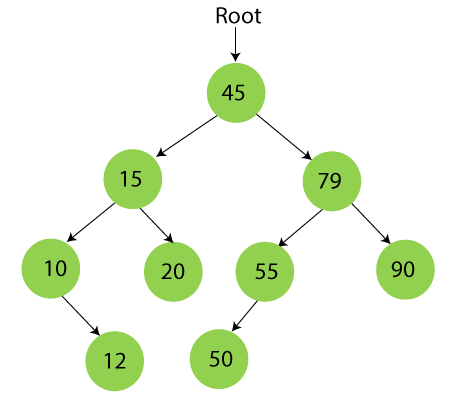
**Step 8 - Insert 20.**

**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.**

****

**Now, the creation of binary search tree is completed. After that, let's move towards the operations that can be performed on Binary search tree.**

**We can perform insert, delete and search operations on the binary search tree.**

**Let's understand how a search is performed on a binary search tree.**

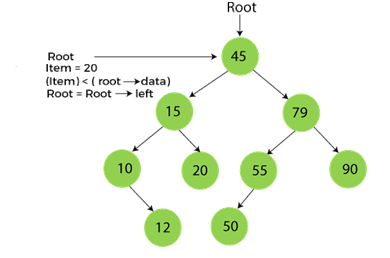
## **Searching in Binary search tree**

**Searching means to find or locate a specific element or node in a data structure. In Binary search tree, searching a node is easy because elements in BST are stored in a specific order. The steps of searching a node in Binary Search tree are listed as follows -**

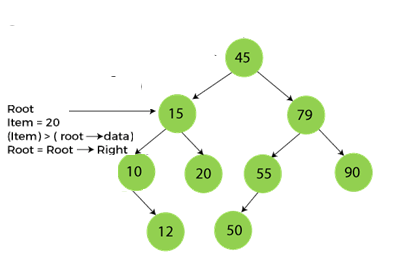
1. **First, compare the element to be searched with the root element of the tree.**
2. **If root is matched with the target element, then return the node's location.**
3. **If it is not matched, then check whether the item is less than the root element, if it is smaller than the root element, then move to the left subtree.**
4. **If it is larger than the root element, then move to the right subtree.**
5. **Repeat the above procedure recursively until the match is found.**
6. **If the element is not found or not present in the tree, then return NULL.**

**Now, let's understand the searching in binary tree using an example. We are taking the binary search tree formed above. Suppose we have to find node 20 from the below tree.**

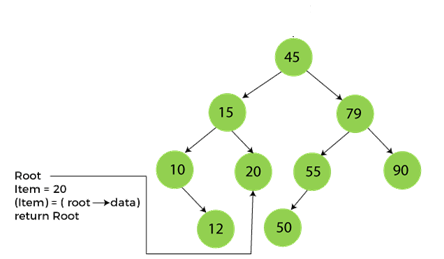
**Step1:**

****

**Step2:**

****

**Step3:**

****

**Now, let's see the algorithm to search an element in the Binary search tree.**

### Algorithm to search an element in Binary search tree

1. **Search (root, item)**
2. **Step 1 - if (item = root → data) or (root = NULL)**
3. **return root**
4. **else if (item < root → data)**
5. **return Search(root → left, item)**
6. **else**
7. **return Search(root → right, item)**
8. **END if**
9. **Step 2 - END**

**Now let's understand how the deletion is performed on a binary search tree. We will also see an example to delete an element from the given tree.**

### Deletion in Binary Search tree

**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 -**

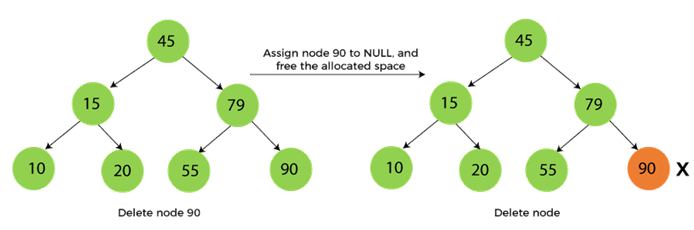
* **The node to be deleted is the leaf node, or,**
* **The node to be deleted has only one child, and,**
* **The node to be deleted has two children**

**We will understand the situations listed above in detail.**

**When the node to be deleted is the leaf node**

**It is the simplest case to delete a node in BST. Here, we have to replace the leaf node with NULL and simply free the allocated space.**

**We can see the process to delete a leaf node from BST in the below image. In below image, suppose we have to delete node 90, as the node to be deleted is a leaf node, so it will be replaced with NULL, and the allocated space will free.**

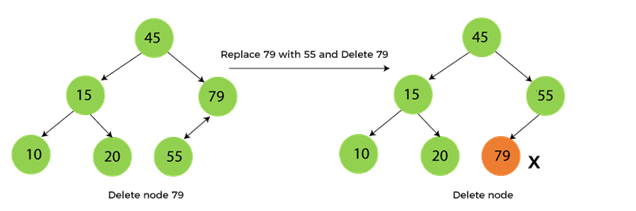
****

**When the node to be deleted has only one child**

**In this case, we have to replace the target node with its child, and then delete the child node. It means that after replacing the target node with its child node, the child node will now contain the value to be deleted. So, we simply have to replace the child node with NULL and free up the allocated space.**

**We can see the process of deleting a node with one child from BST in the below image. In the below image, suppose we have to delete the node 79, as the node to be deleted has only one child, so it will be replaced with its child 55.**

**So, the replaced node 79 will now be a leaf node that can be easily deleted.**

****

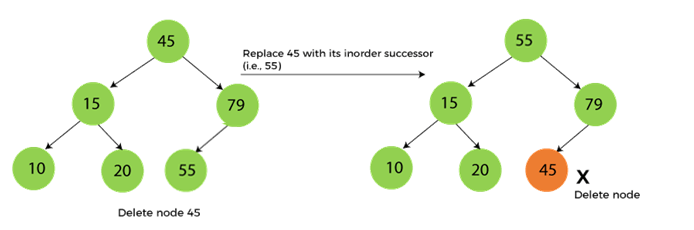
**When the node to be deleted has two children**

**This case of deleting a node in BST is a bit complex among other two cases. In such a case, the steps to be followed are listed as follows -**

* **First, find the inorder successor of the node to be deleted.**
* **After that, replace that node with the inorder successor until the target node is placed at the leaf of tree.**
* **And at last, replace the node with NULL and free up the allocated space.**

**The inorder successor is required when the right child of the node is not empty. We can obtain the inorder successor by finding the minimum element in the right child of the node.**

**We can see the process of deleting a node with two children from BST in the below image. In the below image, suppose we have to delete node 45 that is the root node, as the node to be deleted has two children, so it will be replaced with its inorder successor. Now, node 45 will be at the leaf of the tree so that it can be deleted easily.**

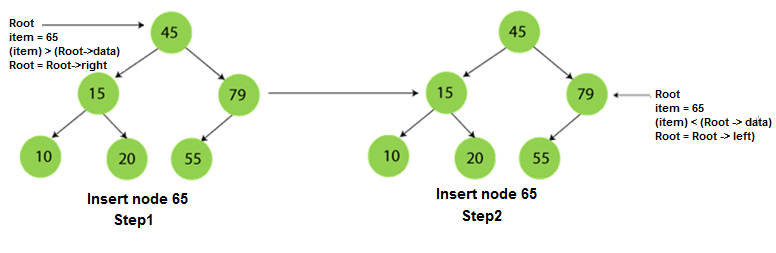
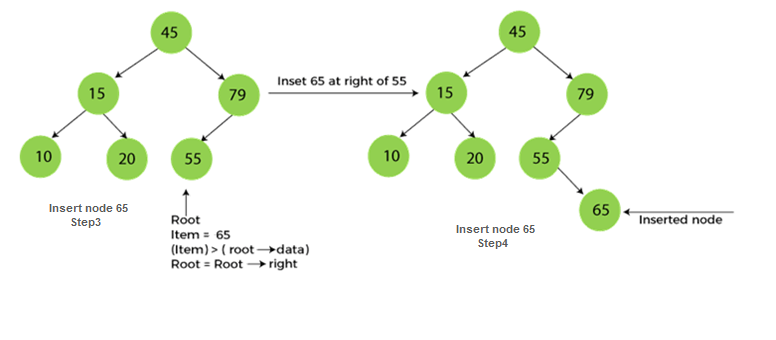
****

**Now let's understand how insertion is performed on a binary search tree.**

### Insertion in Binary Search tree

**A new key in BST is always inserted at the leaf. To insert an element in BST, we have to start searching from the root node; if the node to be inserted is less than the root node, then search for an empty location in the left subtree. Else, search for the empty location in the right subtree and insert the data. Insert in BST is similar to searching, as we always have to maintain the rule that the left subtree is smaller than the root, and right subtree is larger than the root.**

**Now, let's see the process of inserting a node into BST using an example.**

**  
**

### The complexity of the Binary Search tree

**Let's see the time and space complexity of the Binary search tree. We will see the time complexity for insertion, deletion, and searching operations in best case, average case, and worst case.**

### 1. Time Complexity

**Where 'n' is the number of nodes in the given tree.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operations** | **Best case time complexity** | **Average case time complexity** | **Worst case time complexity** |
| **Insertion** | **O(log n)** | **O(log n)** | **O(n)** |
| **Deletion** | **O(log n)** | **O(log n)** | **O(n)** |
| **Search** | **O(log n)** | **O(log n)** | **O(n)** |

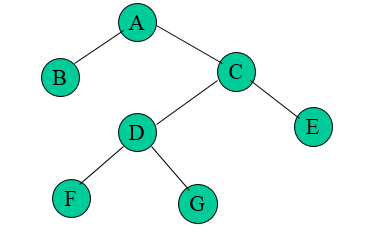
### 2. Space Complexity

|  |  |
| --- | --- |
| **Operations** | **Space complexity** |
| **Insertion** | **O(n)** |
| **Deletion** | **O(n)** |
| **Search** | **O(n)** |

* **The space complexity of all operations of Binary search tree is O(n).**

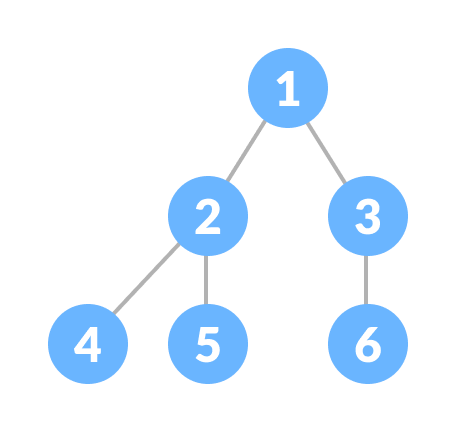
**Strictly binary tree**

**A strictly binary tree with n leaves always contains 2n -1 nodes. If every non-leaf node in a binary tree has nonempty left and right subtrees, the tree is termed a strictly binary tree. Or, to put it another way, all of the nodes in a strictly binary tree are of degree zero or two, never degree one.**

****

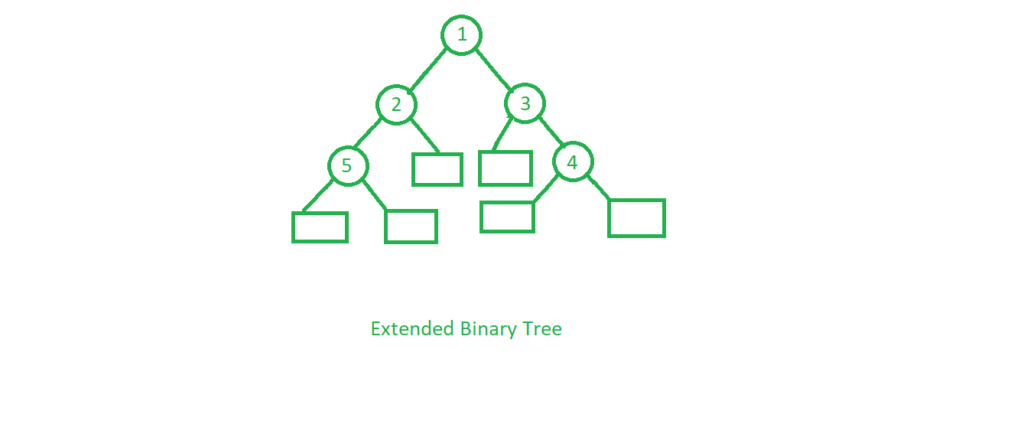
**Complete binary tree**

**A complete binary tree is a binary tree in which all the levels are completely filled except possibly the lowest one, which is filled from the left. A complete binary tree is just like a full binary tree, but with two major differences. All the leaf elements must lean towards the left.**

****

# **Extended Binary Tree**

**Extended binary tree is a type of binary tree in which all the null sub tree of the original tree are replaced with special nodes called external nodes whereas other nodes are called internal nodes**

****

**Here the circles represent the internal nodes and the boxes represent the external nodes.  
Properties of External binary tree**

1. **The nodes from the original tree are internal nodes and the special nodes are external nodes.**
2. **All external nodes are leaf nodes and the internal nodes are non-leaf nodes.**
3. **Every internal node has exactly two children and every external node is a leaf. It displays the result which is a complete binary tree**

# **Tree Traversals (Inorder, Preorder and Postorder)**

**Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.**

**Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree −**

* **In-order Traversal**
* **Pre-order Traversal**
* **Post-order Traversal**

**Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.**

## **In-order Traversal**

**In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.**

**If a binary tree is traversed in-order, the output will produce sorted key values in an ascending order.**

****

**We start from A, and following in-order traversal, we move to its left subtree B. B is also traversed in-order. The process goes on until all the nodes are visited. The output of inorder traversal of this tree will be −**

***D → B → E → A → F → C → G***

### Algorithm

**Until all nodes are traversed −**

**Step 1 − Recursively traverse left subtree.**

**Step 2 − Visit root node.**

**Step 3 − Recursively traverse right subtree.**

## **Pre-order Traversal**

**In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.**

****

**We start from A, and following pre-order traversal, we first visit A itself and then move to its left subtree B. B is also traversed pre-order. The process goes on until all the nodes are visited. The output of pre-order traversal of this tree will be −**

***A → B → D → E → C → F → G***

### Algorithm

**Until all nodes are traversed −**

**Step 1 − Visit root node.**

**Step 2 − Recursively traverse left subtree.**

**Step 3 − Recursively traverse right subtree.**

## **Post-order Traversal**

**In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.**

****

**We start from A, and following Post-order traversal, we first visit the left subtree B. B is also traversed post-order. The process goes on until all the nodes are visited. The output of post-order traversal of this tree will be −**

***D → E → B → F → G → C → A***

### Algorithm

**Until all nodes are traversed −**

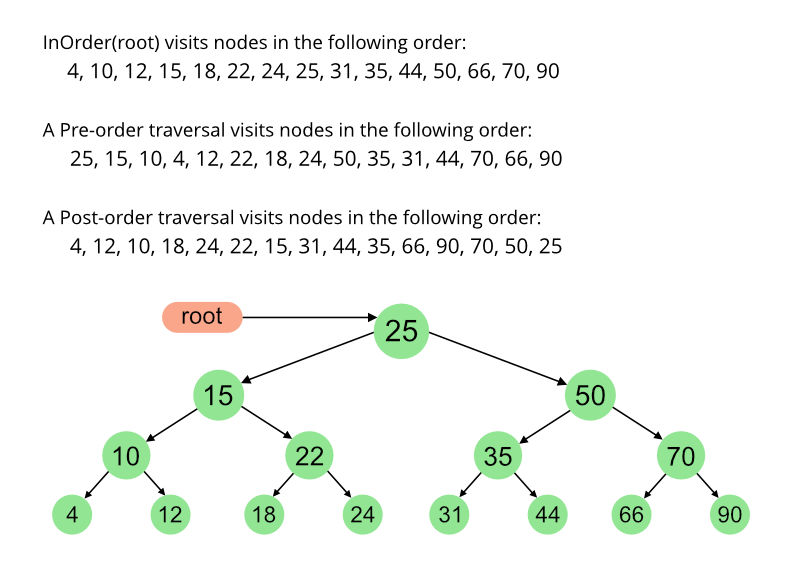
**Step 1 − Recursively traverse left subtree.**

**Step 2 − Recursively traverse right subtree.**

**Step 3 − Visit root node.**

****

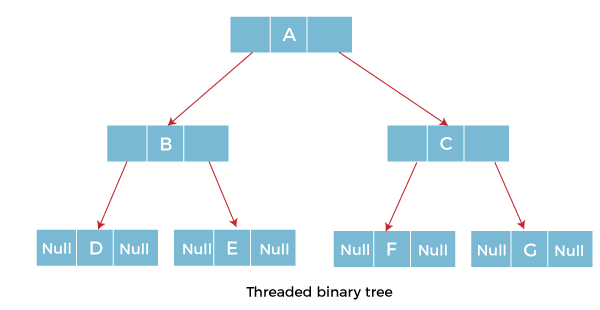
**Depth First Traversals:   
(a) Inorder (Left, Root, Right) : 4 2 5 1 3   
(b) Preorder (Root, Left, Right) : 1 2 4 5 3   
(c) Postorder (Left, Right, Root) : 4 5 2 3 1**

****

# **Threaded Binary Tree**

**What do you mean by Threaded Binary Tree?**

**In the linked representation of binary trees, more than one half of the link fields contain NULL values which results in wastage of storage space. If a binary tree consists of n nodes then n+1 link fields contain NULL values. So in order to effectively manage the space, a method was devised by Perlis and Thornton in which the NULL links are replaced with special links known as threads. Such binary trees with threads are known as threaded binary trees. Each node in a threaded binary tree either contains a link to its child node or thread to other nodes in the tree.**

****

### Types of Threaded Binary Tree

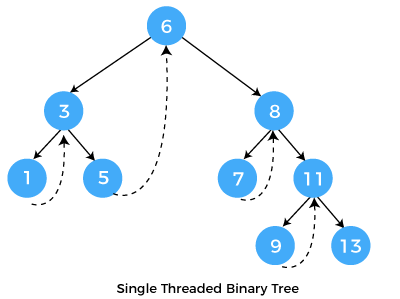
**There are two types of threaded Binary Tree:**

**Nested Structure in C in Hindi**

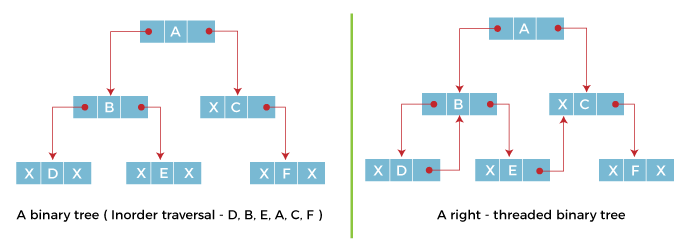
**Keep Watching**

* **One-way threaded Binary Tree**
* **Two-way threaded Binary Tree**

**One-way threaded Binary trees:**

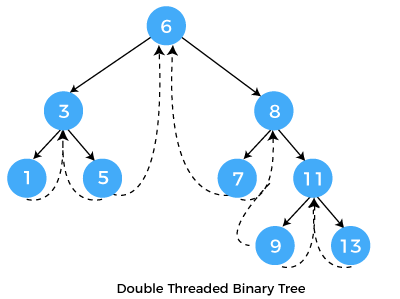
****

**In one-way threaded binary trees, a thread will appear either in the right or left link field of a node. If it appears in the right link field of a node then it will point to the next node that will appear on performing in order traversal. Such trees are called Right threaded binary trees. If thread appears in the left field of a node then it will point to the nodes inorder predecessor. Such trees are called Left threaded binary trees. Left threaded binary trees are used less often as they don't yield the last advantages of right threaded binary trees. In one-way threaded binary trees, the right link field of last node and left link field of first node contains a NULL. In order to distinguish threads from normal links they are represented by dotted lines.**

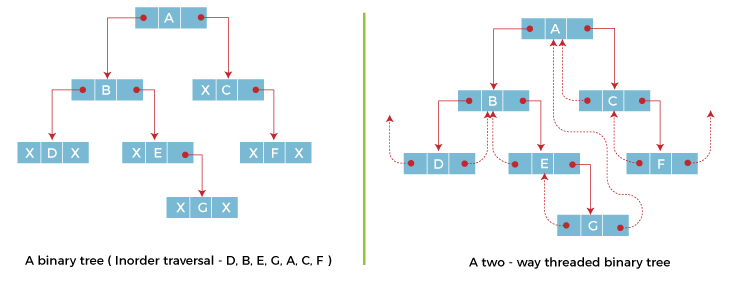
****

**The above figure shows the inorder traversal of this binary tree yields D, B, E, A, C, F. When this tree is represented as a right threaded binary tree, the right link field of leaf node D which contains a NULL value is replaced with a thread that points to node B which is the inorder successor of a node D. In the same way other nodes containing values in the right link field will contain NULL value.**

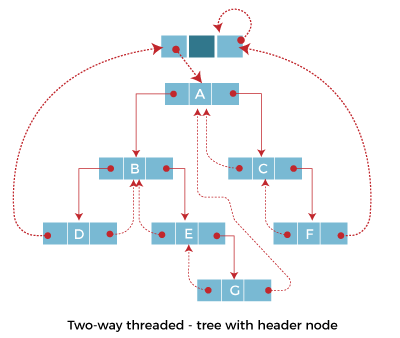
**Two-way threaded Binary Trees:**

****

**In two-way threaded Binary trees, the right link field of a node containing NULL values is replaced by a thread that points to nodes inorder successor and left field of a node containing NULL values is replaced by a thread that points to nodes inorder predecessor.**

****

**The above figure shows the inorder traversal of this binary tree yields D, B, E, G, A, C, F. If we consider the two-way threaded Binary tree, the node E whose left field contains NULL is replaced by a thread pointing to its inorder predecessor i.e. node B. Similarly, for node G whose right and left linked fields contain NULL values are replaced by threads such that right link field points to its inorder successor and left link field points to its inorder predecessor. In the same way, other nodes containing NULL values in their link fields are filled with threads.**

****

**In the above figure of two-way threaded Binary tree, we noticed that no left thread is possible for the first node and no right thread is possible for the last node. This is because they don't have any inorder predecessor and successor respectively. This is indicated by threads pointing nowhere. So in order to maintain the uniformity of threads, we maintain a special node called the header node. The header node does not contain any data part and its left link field points to the root node and its right link field points to itself. If this header node is included in the two-way threaded Binary tree then this node becomes the inorder predecessor of the first node and inorder successor of the last node. Now threads of left link fields of the first node and right link fields of the last node will point to the header node.**

# **AVL Tree**

**AVL Tree is invented by GM Adelson - Velsky and EM Landis in 1962. The tree is named AVL in honour of its inventors.**

**AVL Tree can be defined as height balanced binary search tree in which each node is associated with a balance factor which is calculated by subtracting the height of its right sub-tree from that of its left sub-tree.**

**Tree is said to be balanced if balance factor of each node is in between -1 to 1, otherwise, the tree will be unbalanced and need to be balanced.**

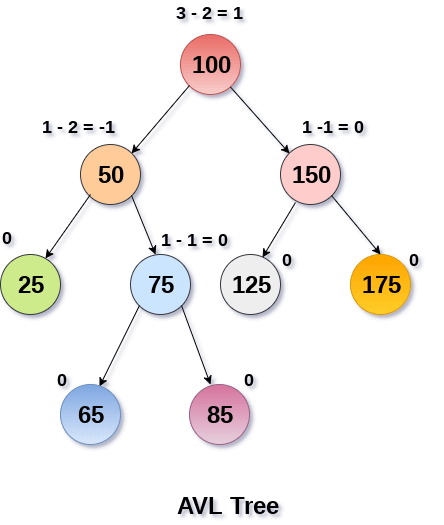
## **Balance Factor (k) = height (left(k)) - height (right(k))**

**If balance factor of any node is 1, it means that the left sub-tree is one level higher than the right sub-tree.Java Try Catch**

**If balance factor of any node is 0, it means that the left sub-tree and right sub-tree contain equal height.**

**If balance factor of any node is -1, it means that the left sub-tree is one level lower than the right sub-tree.**

**An AVL tree is given in the following figure. We can see that, balance factor associated with each node is in between -1 and +1. therefore, it is an example of AVL tree.**

****

## **Complexity**

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Average case** | **Worst case** |
| **Space** | **o(n)** | **o(n)** |
| **Search** | **o(log n)** | **o(log n)** |
| **Insert** | **o(log n)** | **o(log n)** |
| **Delete** | **o(log n)** | **o(log n)** |

## **Operations on AVL tree**

**Due to the fact that, AVL tree is also a binary search tree therefore, all the operations are performed in the same way as they are performed in a binary search tree. Searching and traversing do not lead to the violation in property of AVL tree. However, insertion and deletion are the operations which can violate this property and therefore, they need to be revisited.**

|  |  |  |
| --- | --- | --- |
| **SN** | **Operation** | **Description** |
| **1** | **[Insertion](https://www.javatpoint.com/insertion-in-avl-tree)** | **Insertion in AVL tree is performed in the same way as it is performed in a binary search tree. However, it may lead to violation in the AVL tree property and therefore the tree may need balancing. The tree can be balanced by applying rotations.** |
| **2** | **[Deletion](https://www.javatpoint.com/deletion-in-avl-tree)** | **Deletion can also be performed in the same way as it is performed in a binary search tree. Deletion may also disturb the balance of the tree therefore, various types of rotations are used to rebalance the tree.** |

## **Why AVL Tree?**

**AVL tree controls the height of the binary search tree by not letting it to be skewed. The time taken for all operations in a binary search tree of height h is O(h). However, it can be extended to O(n) if the BST becomes skewed (i.e. worst case). By limiting this height to log n, AVL tree imposes an upper bound on each operation to be O(log n) where n is the number of nodes.**

# **B tree vs B+ tree**

**Before understanding B tree and B+ tree differences, we should know the B tree and B+ tree separately.**

### What is the B tree?

[**B tree**](https://www.javatpoint.com/b-tree)**is a self-balancing tree, and it is a m-way tree where m defines the order of the tree. Btree is a generalization of the**[**Binary Search tree**](https://www.javatpoint.com/binary-search-tree)**in which a node can have more than one key and more than two children depending upon the value of m. In the B tree, the data is specified in a sorted order having lower values on the left subtree and higher values in the right subtree.**

**Properties of B tree**

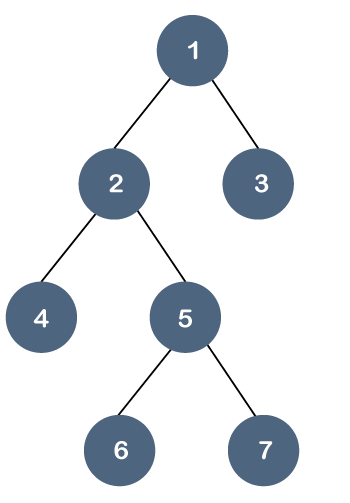
**The following are the properties of the B tree:**

**10 Sec**

**Exception Handling in Java - Javatpoint**

* **In the B tree, all the leaf nodes must be at the same level, whereas, in the case of a binary tree, the leaf nodes can be at different levels.**

**Let's understand this property through an example.**

****

**In the above tree, all the leaf nodes are not at the same level, but they have the utmost two children. Therefore, we can say that the above tree is a**[**binary tree**](https://www.javatpoint.com/binary-tree)**but not a B tree.**

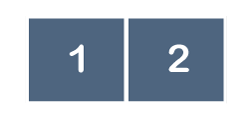
* **If the Btree has an order of m, then each node can have a maximum of m In the case of minimum children, the leaf nodes have zero children, the root node has two children, and the internal nodes have a ceiling of m/2.**
* **Each node can have maximum (m-1) keys. For example, if the value of m is 5 then the maximum value of keys is 4.**
* **The root node has minimum one key, whereas all the other nodes except the root node have (ceiling of m/2 minus - 1) minimum keys.**
* **If we perform insertion in the B tree, then the node is always inserted in the leaf node.**

**Suppose we want to create a B tree of order 3 by inserting values from 1 to 10.**

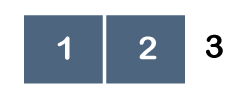
**Step 1: First, we create a node with 1 value as shown below:**

****

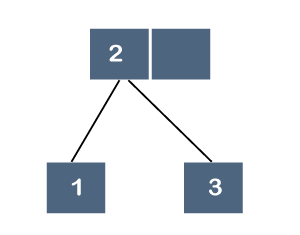
**Step 2: The next element is 2, which comes after 1 as shown below:**

****

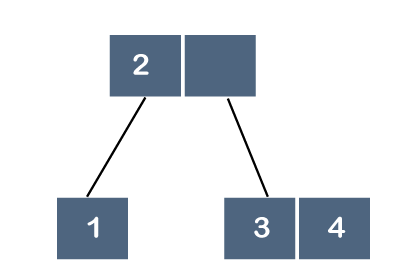
**Step 3: The next element is 3, and it is inserted after 2 as shown below:**

****

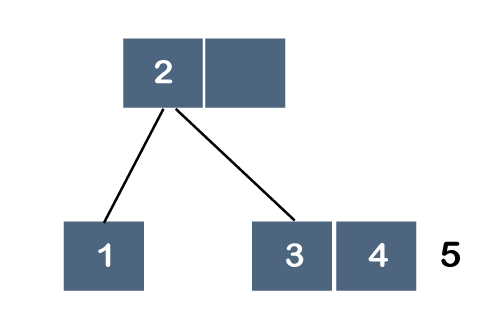
**As we know that each node can have 2 maximum keys, so we will split this node through the middle element. The middle element is 2, so it moves to its parent. The node 2 does not have any parent, so it will become the root node as shown below:**

****

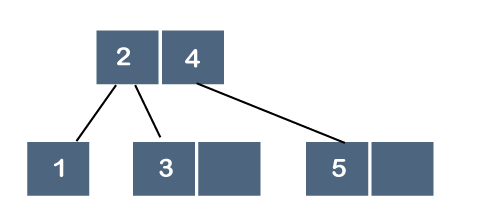
**Step 4: The next element is 4. Since 4 is greater than 2 and 3, so it will be added after the 3 as shown below:**

****

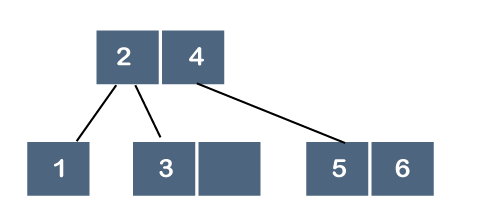
**Step 5: The next element is 5. Since 5 is greater than 2, 3 and 4 so it will be added after 4 as shown below:**

****

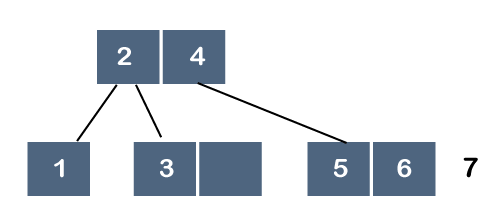
**As we know that each node can have 2 maximum keys, so we will split this node through the middle element. The middle element is 4, so it moves to its parent. The parent is node 2; therefore, 4 will be added after 2 as shown below:**

****

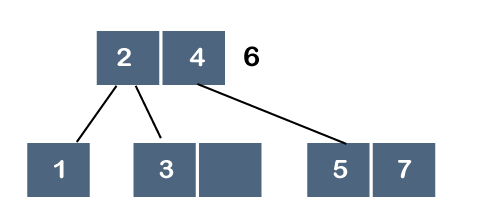
**Step 6: The next element is 6. Since 6 is greater than 2, 4 and 5, so 6 will come after 5 as shown below:**

****

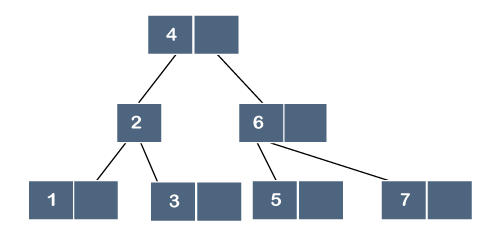
**Step 7: The next element is 7. Since 7 is greater than 2, 4, 5 and 6, so 7 will come after 6 as shown below:**

****

**As we know that each node can have 2 maximum keys, so we will split this node through the middle element. The middle element is 6, so it moves to its parent as shown below:**

****

**But, 6 cannot be added after 4 because the node can have 2 maximum keys, so we will split this node through the middle element. The middle element is 4, so it moves to its parent. As node 4 does not have any parent, node 4 will become a root node as shown below:**

****

### What is a B+ tree?

**The**[**B+ tree**](https://www.javatpoint.com/b-plus-tree)**is also known as an advanced self-balanced tree because every path from the root of the tree to the leaf of the tree has the same length. Here, the same length means that all the leaf nodes occur at the same level. It will not happen that some of the leaf nodes occur at the third level and some of them at the second level.**

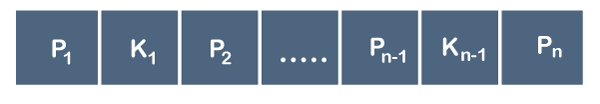
**A B+ tree index is considered a multi-level index, but the B+ tree structure is not similar to the multi-level index sequential files.**

**Why is the B+ tree used?**

**A B+ tree is used to store the records very efficiently by storing the records in an indexed manner using the B+ tree indexed structure. Due to the multi-level indexing, the data accessing becomes faster and easier.**

**B+ tree Node Structure**

**The node structure of the B+ tree contains pointers and key values shown in the below figure:**

****

**As we can observe in the above B+ tree node structure that it contains n-1 key values (k1 to kn-1) and n pointers (p1 to pn).**

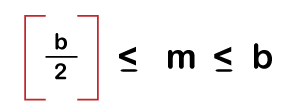
**The search key values which are placed in the node are kept in sorted order. Thus, if i<j then ki<kj.**

**Constraint on various types of nodes**

**Let 'b' be the order of the B+ tree.**

**Non-Leaf node**

**Let 'm' represents the number of children of a node, then the relation between the order of the tree and the number of children can be represented as:**

****

**Let k represents the search key values. The relation between the order of the tree and search key can be represented as:**

**As we know that the number of pointers is equal to the search key values plus 1, so mathematically, it can be written as:**

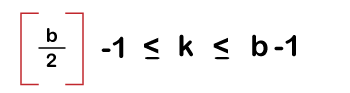
**Number of Pointers (or children) = Number of Search keys + 1**

**Therefore, the maximum number of pointers would be 'b', and the minimum number of pointers would be the ceiling function of b/2.**

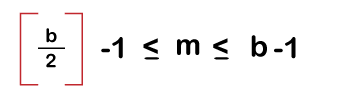
**Leaf Node**

**A leaf node is a node that occurs at the last level of the B+ tree, and each leaf node uses only one pointer to connect with each other to provide the sequential access at the leaf level.**

**In leaf node, the maximum number of children is:**

****

**The maximum number of search keys is:**

****

**Root Node**

**The maximum number of children in the case of the root node is: b**

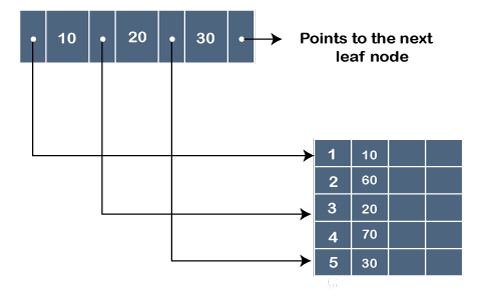
**The minimum number of children is: 2**

**Special cases in B+ tree**

**Case 1: If the root node is the only node in the tree. In this case, the root node becomes the leaf node.**

**In this case, the maximum number of children is 1, i.e., the root node itself, whereas, the minimum number of children is b-1, which is the same as that of a leaf node.**

**Representation of a leaf node in B+ tree**

****

**In the above figure, '.' represents the pointer, whereas the 10, 20 and 30 are the key values. The pointer contains the address at which the key value is stored, as shown in the above figure.**

**Example of B+ tree**

****

**In the above figure, the node contains three key values, i.e., 9, 16, and 25. The pointer that appears before 9, contains the key values less than 9 represented by ki. The pointer that appears before 16, contains the key values greater than or equal to 9 but less than 16 represented by kj. The pointer that appears before 25, contains the key values greater than or equal to 16 but less than 25 represented by kn.**

### Differences between B tree and B+ tree

|  |  |
| --- | --- |
| **B tree** | **B+ tree** |
| **In the B tree, all the keys and records are stored in both internal as well as leaf nodes.** | **In the B+ tree, keys are the indexes stored in the internal nodes and records are stored in the leaf nodes.** |
| **In B tree, keys cannot be repeatedly stored, which means that there is no duplication of keys or records.** | **In the B+ tree, there can be redundancy in the occurrence of the keys. In this case, the records are stored in the leaf nodes, whereas the keys are stored in the internal nodes, so redundant keys can be present in the internal nodes.** |
| **In the Btree, leaf nodes are not linked to each other.** | **In B+ tree, the leaf nodes are linked to each other to provide the sequential access.** |
| **In Btree, searching is not very efficient because the records are either stored in leaf or internal nodes.** | **In B+ tree, searching is very efficient or quicker because all the records are stored in the leaf nodes.** |
| **Deletion of internal nodes is very slow and a time-consuming process as we need to consider the child of the deleted key also.** | **Deletion in B+ tree is very fast because all the records are stored in the leaf nodes so we do not have to consider the child of the node.** |
| **In Btree, sequential access is not possible.** | **In the B+ tree, all the leaf nodes are connected to each other through a pointer, so sequential access is possible.** |
| **In Btree, the more number of splitting operations are performed due to which height increases compared to width,** | **B+ tree has more width as compared to height.** |
| **In Btree, each node has atleast two branches and each node contains some records, so we do not need to traverse till the leaf nodes to get the data.** | **In B+ tree, internal nodes contain only pointers and leaf nodes contain records. All the leaf nodes are at the same level, so we need to traverse till the leaf nodes to get the data.** |
| **The root node contains atleast 2 to m children where m is the order of the tree.** | **The root node contains atleast 2 to m children where m is the order of the tree.** |