UNIT - III

Tree Structure: Concept and terminology, Types of tress, Binary search tree, inserting, deleting and searching into binary search tree, implementing the insert , search and delete algorithms, tree traversals, Huffman's algorithm.

UNIT – III

Chapter ▸ 4

Tree Structure

4.1 Introduction

A tree is recursively (locally) defined as a set of one or more node where one node define as root node(parent node) and others are child of root node. Mean, a tree (T)is a collection of nodes such that nodes have a parent-child relationship .

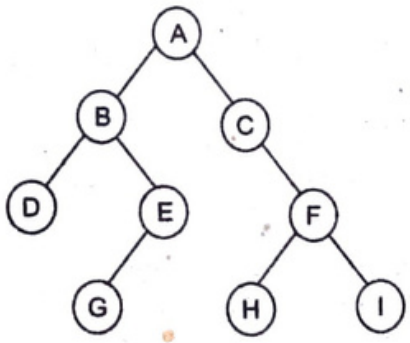


Figure 4.1

Here, A node is called as root node (parent node) and remaining nodes are child node.

4.1.1 Basic Terminology

Root node: The root node is the topmost node in the tree. In empty tree , root node is null.

Leaf-node: A node that have no children is called Leaf node or Terminal node. In fig. 4.1,G,H,I are leaf node

Path: A sequence of consecutive edges is called a path. For e.g. the path from the root node A to D node is given as A,B,D.

Ancestor node: A node that is connected to all lower-level nodes is called Ancestor node. The root node does not have any ancestors. In fig 4.1 node A does not have any ancestor.

Descendant node: A node is any successor node on any path from the node to a leaf node. Leaf node has no descendants. In the tree given in fig 4.1, node B,E,G are the descendent of node A. Node G,H,I have no descendents.

Sub-tree: If the root node is not is not NULL, then the trees T_1,T_2 and T_3 are called sub-trees of root node.

Level: Every node in the tree is assigned a level number in such a way that the root node is at level 0,children of the root node are at level number 1, so on.

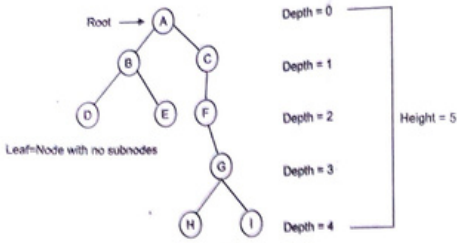


Figure 4.2

Height: The height of node is the number of edges on the longest downward path between that node and a leaf.

Depth: The depth of node is the number of edges from the node to the tree's root node.

Note: Level is depth plus 1

Degree: Degree of a node is equal to the number of children that a node has. The degree of a leaf node is zero.

In-degree: In-degree of a node is the number of edges arriving at the node. The node has 0 in-degree.

Out-degree: Out-degree of a node is the number of edges leaving that node. The node A has 0 in degree. The node D,E,H,I has 0 out degree.

4.2 Types Of Trees

Trees are of following 6 types:-

1. General trees
2. Forests
3. Binary trees
4. Binary Search tree
5. Expression tree
6. Tournament tree

4.2.1 General Trees

General tree are data structure that store elements hierarchically. The top node of a tree is the root node and each node except the root , has a parent. A node in a general tree (except the leaf nodes) may have zero or more sub-trees. General tree have a 3 sub-tree per node are called ternary nodes. However, the number of sub-trees for any node may be variable. For example, a node can have 1 sub-tree, whereas some other node can have 3 sub-trees.

Although general trees can be represented as ADTs , there is always a problem when another sub-tree is added to a node that already has the maximum number of sub-trees attached to it. Even the algorithm for searching, traversing, adding and deleting nodes become much more complex as there are not just two possibilities for any node but multiple possibilities.

To overcome the complexities of a general tree ,it may be represented as a graph data structure , thereby losing many of the advantages of the tree processes. Therefore a better option is to convert general trees into binary trees.

A general tree when converted to a binary tree may not end up being well formed or full but the advantages of such a conversion enable the programmer to use the algorithm for processes that are used for binary trees with minor modifications.

4.2.2 Forests

A forest is a set of $n \geq 0$ disjoint trees. A set of disjoint trees are obtained by removing the root and the edges connectiong the root node to nodes at level 1.

A forest can also be defined as an ordered set of zero or more general trees. While a general tree must have a root, a forest on the other hand may be empty because by definition it is a set, and sets can be empty.

A forest can be converted into a tree by adding a single node as the root node of the tree. Similarly , general tree into a forest by deleting the root node of the tree. A forest can also be represented by a binary tree. Fig 5.3(a) shows a forest and fig. 5.3(b) shows the corresponding tree. Obviously, an empty forest can be represented by the empty binary tree.

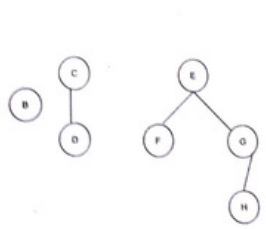


Figure 4.3 (a) Shows a forest

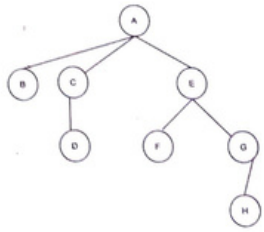


Figure 4.3 (b) Shows Corresponding Tree

4.2.3 Binary Tree

Binary tree is defined as collection of nodes. In a binary tree the topmost element is called the root node has 0,1 or at most 2 children. In other words binary tree is either empty(root=null) or consists of a node called root node with two binary tree called left sub-tree and right sub-tree. A node that has zero children is

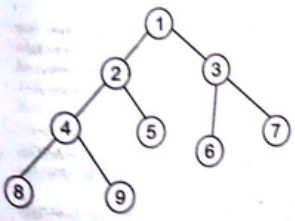


Figure 4.4(a)

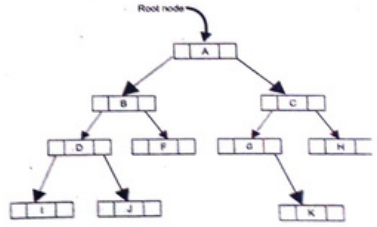


Figure 4.4(b) Linked Datastructure of Binary Tree

called leaf node or terminal node.Binary tree is linked data structure containing nodes more than one self. referenced field. A binary tree made of nodes, where each node contain a "left" reference(left pointer) and "right" reference(right pointer).The root element is pointed by 'root' pointer. Fig. 5.5(a) shows the binery tree and fig. 5.4(b) shows the linked structure of Binery tree.

4.2.3.1 Terminology

Parent: If a node in binary tree has a left successor and right successor , then node is called parent of left and right successor. Every node other than root node has a parent.

Level number: Every node in a binary tree is assigned a level number.The rrot node is defined at Level 0. The left child and righr child of root node is defined at Level 1, so on. Every node is at one level higher than its parents. So all child nodes are defined to have level number as parent's level number + 1.

Degree of a node: Number of children of a node is called degree of that node. The degree of leaf node is zero because leaf node is terminal node so no child and no degree.

Sibling: All nodes at same level and share same parent are called as siblings. For example in fig. 5.4(a), nodes 2 and 3; 4 and 5; 6 and 7; 8 and 9; are siblings.

Leaf node: A node that has no children is called as leaf node or terminal node.

Similar binary trees: Suppose two binary tree T T' are similar binary tree if both have same structure

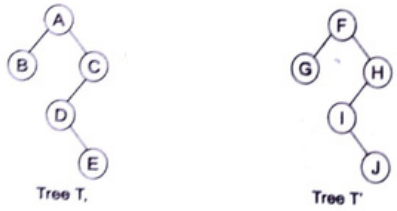


Figure 4.5 Similar Binary Tree

Path: A sequence of consecutive edges.

Depth: The depth of a node is the number of edges from root to that node. The depth of root node is zero.

Height of Tree: The height of a node is the number of edges from the node to the deepest leaf. The height of tree is the height of root.A tree with only a root node has a height of 1. A binary tree of height h has at least h nodes and atmost 2^h-1 nodes. This is because every level will have at least one node and can have at most 2 nodes. So, if every level has two nodes then a tree with height h will have at most 2^h-1 nodes as at level 0,there is only one element called the root.The height of binary tree with n nodes is at least $\log_2(n+1)$ and at most n.

In-degree/Out-degree: In-degree of a node is the number of arriving edges at node. Root node has no in-degree. Whereas out-degree of a node is the number of leaving edges from that node. Leaf node of a tree has no out-degree .

Binery trees are commonly used to implement binary search trees, expression trees tournament trees and binary heaps.

4.2.3.2 Complete Binary Trees

A complete binary tree has two property:

1. In complete binary tree ,every level, except the last level ,is completely filled with nodes.
2. All node appear as far left as possible.

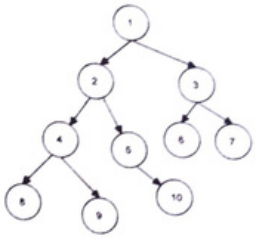


Figure 4.6 Complete Binary Tree

In fig. 4.6, tree has exactly 10 nodes. We have purposly labelled from 1 to 10 so that it is easy for the reader to find the parent node, the right child node, and left child node of the given node. The formula given as if k is a parent node, them its left child can be calculation as

$$2 \times k$$

and its right child can be calculated as

$$2 \times k+1$$

For example the left child of node 4 is 8 (2×4)
and right child of node 4 is 9 (2×4+1)
similarly , the parent of the node k can be calcuted as

$$\lfloor k/2 \rfloor$$

like the parent node of node 4 is $\lfloor 4/2 \rfloor = 2$. Yes, it is
The Height of a tree T_n in having exactly n node is given as

$$H_n = \lfloor \log (n+1) \rfloor$$

4.2.3.3 Full Binary Tree

A full binary tree (sometimes proper binary tree or 2-tree)is a tree in which every node other than the leaves has two leaves.

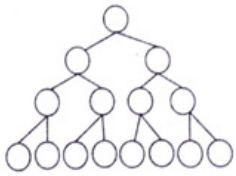


Figure 4.7 Full Binary Tree

4.2.3.4 Extended Binary Tree

A binary tree T is said to be an extended binary tree (or a 2-tree) if each node in the tree has either no child or exactly two children .Nodes having two children are called internal nodes and the nodes having no children are called external nodes.

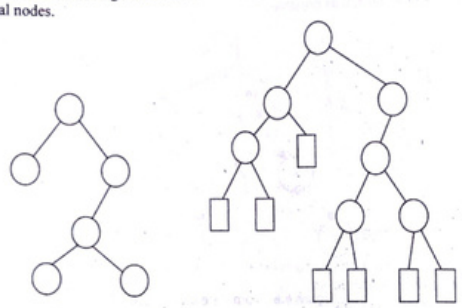


Figure 4.8 (a) & (b)

In fig. 4.8, internal nodes are represented using and external nodes are represented as squares.

To convert a binary tree into an extended tree, every empty sub-tree is replaced by new node. The original nodes in tree are internal node and the new node are added called external nodes.

4.2.4 Binary Search Tree

Binary search tree is also ordered binary tree or sorted binary tree.Binary search tree has a property that ,each internal node,say X store an element such that the element stored in left subtree of of X are less than or equal to X and element of right subtree of X are greater than or equal to X. (Binery search tree will discussed later)
Fig. 4.9 Binery search tree

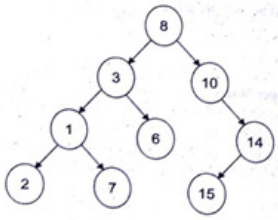


Figure 4.9 Binary Search Tree

4.2.5 Expression Tree

Bianary tree has an application to store algebraic expressions in it. For example consider the algebraic expression given as:

Exp= (A+B)*((C-D)/(E^F))

This expression can be represented using a binary tree as shown in fig.

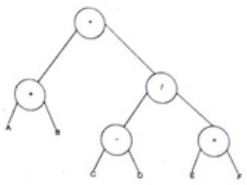


Figure 4.10 Expression Tree

4.2.6 Tournament Trees

Tournament tree also called a selection tree where each external node is a player and each internal node represents the winner of the match played between the players represented by its children .These tournament trees also called as winner trees because they are being used to record the winner at each level, We can also have a loser tree that records the loser at each level.

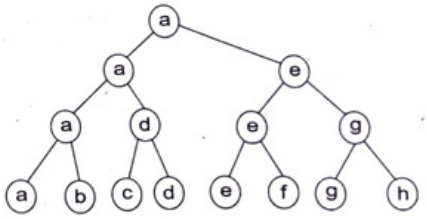


Figure 4.11 Tournament Tree

Consider the tournament tree as shown above. There are 8 players in total whose names are represented using a,b,c,d,e,f,g and h. In round 1, a and b ; c and d ; e and f; and finally g and h play against each other. In round 2, the winners of round 1 ,i.e. a,d,e and g play against each other. In round 3, the winners of round 2, a and e play against each other. Whosoever wins is declared the winner . In the tree, the root node a specifies the winner.

4.3 Traversal of Trees

Traversing a binary tree means a process of visiting each node in the tree exactly once in a systematic order. Unlike, linear data structure where traversing done in sequential order, trees are non-linear data structure which can be travers in different ways. There are different algorithms for tree traversal. During traversing each node

of tree, these nodes are stored in data structure such as Stack or Queue. Traversing can be done in two major way, first ,one does go down first (depth first search) and second, one does go across level(breadth first search).

4.3.1 Depth-First-Search

Tree can be searched in three ways, pre-order, in-order and post-order. These searching technique refered to as depth-first search.

4.3.1.1 Pre-order Traversal

To traverse a non-empty binary tree in pre-order ,the following operations are performed recursively at each node. The algorithm works as:-

1. Visting the root node.
2. Traversing the left subtree, and finally
3. Traversing the right subtree.

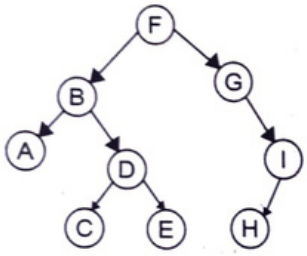


Figure 4.12 Pre-order Traversal

Consider the fig.4.12 .The pre-order traversal of the tree is given asF,B,A,D,C,E,G,I,H. As we study that in Pre-order traversing method first root node is traversed .In figure 4.12 is root node. Move to left subtree whose root is B , so traverse it . Next left node is A then move to right node which is D . D node has a left node i.e. C and right node is E. Now move to right subtree of root node F , G is traversed then I then H.

Algorithm for pre-order traversal

```
Step1:    Repeat steps 2 to 4 while TREE !=NULL
Step2:    write TREE.->DATA
Step3:    PREORDER(TREE->LEFT)
Step4:    PREORDER(TREE->RIGHT)
          [END OF LOOP]
Step5:    END
```

Pre-order traversal algorithm are used in extract a prefix notation forom an expression tree. For e.g. Consider the expression given below when we traversal the elements of 0 tree of fig. 4.10 using pre-order traversal alglorithm the expression that we get is in prefix

* + A B / - C D ^ E F (from fig. 4.10)

4.3.1.2 In-Order Traversal

To traverse a non-empty binary tree in in-order ,the following operations are performed recursively at each node. The algorithm works as:-

1. Traversing the left sub-tree.
2. Visiting the root node , finally
3. Traversing the right sub-tree.

Consider the fig 4.12 The in-order traversal of the tree is given as A,B,C,D,E,F,G,H,I

As we study in In-order traversing method that first left sub-tree is traversing , here F as a root node has left sub-tree B whose left node is A which is first traversed. A is stored in Stack . Acc. to traversing method next to visit root node of A which is B , stored it in Stack. Next after traversing root node move to right sub-tree of B . Left node of D is C go to stack then D itself , and last right node of D i.e E stored in stack.In this way Left sub-tree of F is traversed , now turn to visit root node,i.e F . Now move to right sub tree of F and visiting in above stated manner.

Algorithm for in-order traversal

```
Step1:    Repeat steps 2 to 4 while TREE !=NULL
Step2:    INORDER(TREE->LEFT)
Step3:    write TREE.->DATA
Step4:    INORDER(TREE->RIGHT)
          [END OF LOOP]
Step5:    END
```

In-Order traversal algorithm is usually used to display the elements of Binary Search tree. Here all the elements with a value lower than a given value accssed before the elements with a higher value.

4.3.1.3 Post-order Traversal

To traverse a non-empty binary tree in pre-order ,the following operations are performed recursively at each node. The algorithm works as:-

1. Traversing the left subtree
2. Traversing the right sub-tree ,and finally
3. Visiting the root node.

Consider the fig 4.12 The Post-order traversal of the tree is given as A,C,E,D,B,H,I,G,F

First have to travers left subtree of F which is root with B node. B node has left node A which is traversel first and stored in stack. Acc. to algo. above to the right child of B which is D. D has a left child C which is stored in stack and then E has right child. After that D is stored then B. Now move to the right subtree of F which has child G. G has a I node as a parent of H. This time H is stored continue as I, G, F. So, A, C, E, D, B, H, I, G, F sequence we get.

Algorithm for post-order traversal

```
Step1:    Repeat steps 2 to 4 while TREE !=NULL
Step2:    POSTORDER(TREE->LEFT)
Step3:    POSTORDER(TREE->RIGHT)
Step4:    Write TREE->DATA
          [END OF LOOP]
Step5:    END
```

4.3.2 Breadth-First Search

It is also called as Level–order traversal. In this all nodes at a level are accessed before going to the next level.

Consider the fig. 4.12. Traversal order is F,B,G,A,D,I,C,E,H. As above stated that all nodes of Level 0 is visited first. Next all nodes of Level 1 are visited(B,G). Then Level 2(A,D,I) and last C, E, H. (level 3)

4.4 Binary-Search Tree

We have already discussed binary trees.A binary search tree , is an ordered binary tree in which the nodes are arranged in an order. In a binary-search tree , all nodes in the left sub-tree have a value less than that of the root node. Correspondingly, all the nodes in right sub-tree have a value either *greater than or equal to root node*. The same rule applicable to every sub-tree in the tree. In binary search tree every node contain one value and two pointers left and right, which point to the node's left and right sub-trees, respectively. As shown in fig. 4.13

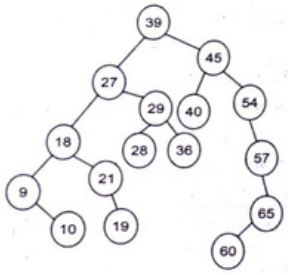


Figure 4.13

Look at the figure. The root node is 39. The left sub-tree of tree is 27 in value which is less than 39, the right node of root node is 45 which is greater than root node. When we jump on left sub-tree whose root node is 27 . The left node of 27 are 10,9,19,21 all are less than 27. The right node of 27 are 28,29,36 all are greater than 27. Similiarly when saw the right subtree of node 39 , whose root node is 45. The left nodes of 45 are 40 which is less than 45, the right nodes of 45 are 54,59,65 and 60 all are greater than 45.

Such type of trees are called binary search tree.

When data changes rapidly , in such cases, Binary search tree speed up the insertion and deletion operations in $O(\log_2 n)$ time. Binary search tree is efficient data structure especially when compared with sorted linear arrays and linked lists. In sorted array, searching can be done in O(log n) time, but insertions and deletions are quite expensive .In contrast, inserting and deleting elements in a linked list is easier , but searching for an element is done in O(n) time.

However, in worst case , a binary search tree will take O(n)time to search for an element.The worst case occur when the tree is linear order. as given in fig. 4.14

(a) Left Skewed,
(b) Right skewed binary search tree.

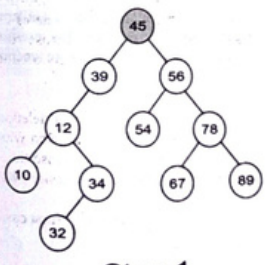
Figure 4.14 (a) Left Skewed (b) Right Skewed

4.4.1 Operations on Binary Search Tree

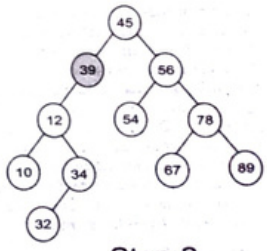
Here we discuss the list of operations which can be performed on Binary search tree.

4.4.1.1 Searching for a Node in Binary Search Tree

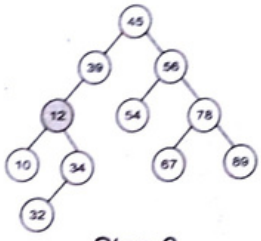
Searching means to find a value whether present or not. The searching process starts with root node. First it check either the tree is empty or not. If Tree is empty it means value which we searching is not be present. So searching algorithm terminates. However , if tree have a nodes, then the search function checks to see if the key value of the current node is equal to the value of the current node is equal to the value is searched. If not, it checks if the value to be searched for is less than the value of the current node , in which case it should be recursively called on the left child. In case the value is greater than the value of the current node it should be recursively called on the right node.



Step-1



Step-2



Step-3

Figure 4.15 Searching a node with value 12 in the given binary search tree.

Look at figure. The figure shows how a binary tree is searched to find a specific node.First see how the tree traversed to find a node with a value 12.

Algorithm to search for a given node

```
SearchElement (TREE,VAL)
Step1: If TREE-> DATA = VAL OR TREE=NULL
    RETURN TREE
  ELSE
   IF VAL < TREE -> DATA
     RETURN searchElement(TREE->LEFT,VAL)
   ELSE
     RETURN searchElement(TREE->RIGHT,VAL)
     [END OF IF]
     [END OF IF]
  Step 2: END
```

4.4.1.2 Inserting a New node in a Binary Search Tree

Adding a new node in a tree should not violet the property of binary search tree.The initial code is similar to search function. This is because we first to find the location where the new node has to be insert .The insertion function changes the structure of the tree. Therefore when the insert function is called recursively the function should return the new tree pointer.

Algorithm to insert a new node in binary search tree

```
INSERT (TREE, VAL)
Step1: IF TREE= NULL
     Allocate memory for TREE
     SET TREE -> DATA=VAL
     SET TREE->LEFT=TREE->RIGHT =NULL
     ELSE
     IF VAL < TREE->DATA
       Insert (TREE->LEFT, VAL)
   ELSE
     Insert (TREE->RIGHT,VAL)
   [END OF IF]
   [END OF IF]
```


In step1 of algorithm, insert function checks if current node of tree is NULL.If it is NULL, the algorithm simply add a new node else it looks at the current node's value and then recurs down the left or right sub-tree.

If the current node's value is less than that of the new node , then the right sub-tree is traversed .else the left sub-tree is traversed . The insert function continues moving down the levels of a binary tree until it reaches a leaf node .The new node is added by following the rules of the binary search trees. That is if the new node 's value is greater than that of the parent node, the new node is inserted in the right sub-tree ,else it is inserted in the left sub-tree.

The insert function require time proportional to the height of the tree in the worst case .It takes O(log n) time to execute in the average case and O(n) time in the worst case .

4.4.1.3 Deleting a Node from Binary Search Tree

The delete function used to delete a node from binary search tree. However utmost care should be taken that properties of binary search tree should not be violated. Here we discuss three cases how node is deleted from a binary tree.

Case 1. Deleting A Node That Has No Children

Look at the binary search tree in fig.4.16

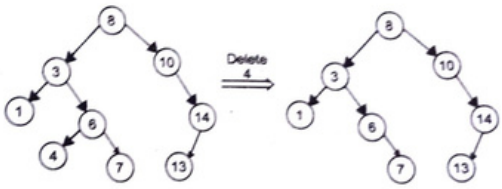


Figure 4.16 Delete node 4 from the binary search tree

If we have to delete node 4 , we can easily remove it without any issue.

Case 2: Deleting A Node With One Child

To handle such case, the node's child is set to as the child of node's parent. In other words, replace the node with its child. Now, if node is left child of its parent , the node's child becomes the left child of the node's parent. Correspondingly, if the node is the right child of its parent , the node's child becomes the right child of the node's parent.

Fig. shows how deletion of 14 is handle

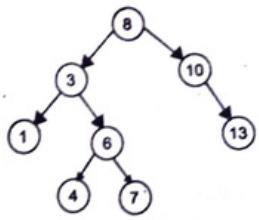
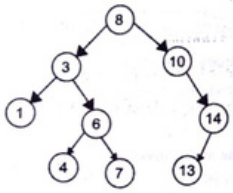


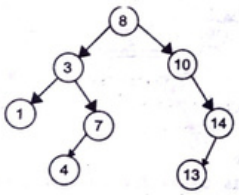
Figure 4.17 Delete node 14 from the Binary search tree.

Case 3: Deleting A Node With Two Children

To handle such case, replace the node's value with its *in-order predecessor* (largest value in the left sub-tree) or *in-order successor* (the smallest value in the right sub-tree). The in-order predecessor and in-order successor can be deleted using any of the above cases.
fig. shows how deletion of 6 is handle.



Step (2)



Step (3)

Figure 4.18 Step 1 & 2 Deleting A Node With Two Children

Algorithm to delete a node from a binary search tree

```
Delete (TREE,VAL)
Step1: IF TREE =NULL
     Write "VAL not found in the tree"
   ELSE IF VAL< TREE -> DATA
     DELETE (TREE->LEFT ,VAL)
   ELSE IF VAL > TREE->DATA
     DELETE (TREE->RIGHT ,VAL)
   ELSE IF TREE->LEFT  AND TREE->RIGHT
     SET TEMP=findLargestNode (TREE->LEFT)
     SET TREE-> DATA=TEMP->DATA
     DELETE (TREE->LEFT, TEMP->DATA)
  ELSE
     SET TEMP =TREE
     IF TREE->LEFT=NULL AND TREE->RIGHT =NULL
     SET TREE=NULL
  ELSE IF TREE ->LEFT != NULL
```



```
      SET TREE=TREE->LEFT
    ELSE
      SET TREE=TREE->RIGHT
    [END OF IF]
    FREE TEMP
     [END OF IF]
Step 2: END
```

The delete function requires time proportional to the height of the tree in the worst case. It takes O(logn) time to execute in the average case and Ω(n) time in the worst case.

4.4.1.4 Determining the Height of a Binary Search Tree

To determine the height of BST, we have to calculate the height of right sub-tree and left sub-tree whichever is greater, 1 is added to it.

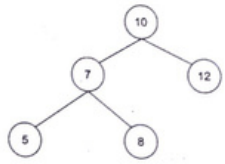


Figure 4.19

For example. In given figure, height of left sub-tree is greater than than the righr sub-tree.
The height of tree=(height of left sub-tree)+1= 2+1=3

Algorithm to find the height of Tree

```
Height (TREE)
Step1: IF TREE=NULL
         Return 0
     ELSE
     SET LeftHeight=Height(TREE->LEFT)
     SET RightHeight=Height(TREE->RIGHT)
     IF LeftHeight.RightHeight
        Return LeftHeight+1
     ELSE
        Return RightHeight+1
    [END OF IF]
    [END OF IF]
  Step2: END
```

In step 1 we check it the current node of tree=NULL.If the condition is true, then 0 is returned to the calling code.otherwise for every node, we redursively call the algorithm to calculate the height of its left sub-hight of the tree at that node is given by adding 1 to the height of the left sub-tree or the height or the height of right sub-tree, which ever is greater.

4.4.1.5 Determinig The Number Of Nodes

To calculate total number of internal nodes and non-leaf nodes, we count the number of internal node in the left sub-tree and right sub-tree and add 1 to it(1 is added for the root node).

Number of nodes= total nodes (left sub-tree)+total node (right sub-tree)+1

In given figure. 4.19 the number of nodes in left sub-tree are 3 and 1 in right sub-tree. So total number of nodes are=3+1+1=5.

Algorithm to calculate number of nodes

```
totalNodes   (TREE)
Step 1: IF TREE=NULL
      RETURN 0
   ELSE
     RETURN totalNodes(TREE->LEFT)
       +totalNodes(TREE->RIGHT)+1
   [END OF IF]
Step 2:  END
```

4.4.1.6 Determining the Number of Internal Nodes

To calculate the total number of internal nodes or non-leaf nodes, we count the number of internal nodes in the left sub-tree and right sub-tree and add 1 to it (1 is added for the root node)

Number of internal node = total internal node (left sub-tree)+ total internal node (right sub-tree)

consider fig. 4.20 the total number of internal nodes in the tree can be calculated as

Total internal nodes of left sub-tree = 0

Total internal nodes of right sub-tree = 3

Total internal nodes of tree = (0+3)+1=4

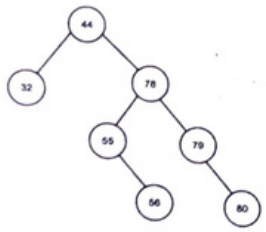


Figure 4.20 Binary Search Tree

Algorithm to calculate the total number of internal nodes in BST total Internal Nodes (Tree)

```
   Step 1: IF TREE = NULL
      Return 0
   [END OF IF]
   IF TREE → LEFT = NULL AND TREE → RIGHT = NULL
      Return 0
   ELSE
      Return total internal Nodes ( TREE → LEFT)+ total Internal Nodes (TREE →
RIGHT)+1
   [END OF IF]
   STEP 2: END
```


4.4.1.7 Detrmining the Number Of External Nodes

If Tree is empty or Null, then the number of external nodes will be zero. If tree have nodes, we add the number of external nodes in the left sub-tree and the right sub-tree.If tree has only one node , in such case external node will be one. Consider the tree in fig. 4.20. The total number of External nodes in the tree can be calculated as

Total external node in left sub-tree = 1

Total external node in right sub-tree = 2

Total external node in tree = 1+2 = 3

Algorithm to calculate total number of external nodes

```
totalExternalNodes(TREE)
step 1: IF TREE=NULL
      RETURN 0
   ELSE IF TREE->LEFT=NULL AND TREE->RIGHT=NULL
      RETURN 1
     ELSE
      RETURN totalExternalNodes(TREE->LEFT)+
      totalExternalNodes(TREE->RIGHT)
     [END OF IF]
Step 2: END
```

4.5 AVL Trees

An AVL Tree is another balanced binary search tree.Named after, Adelson, Velskii and Landis,they were the first dynamically balanced binary search trees to be proposed. In an AVL trees the heights of two sub-trees of a node may differ by at most one . Due to this property, the AVL trees is also known as **Height-balanced tree.** The key advantages of using an AVL tree is that it takes O(log n) time to perform search ,insert, and delete operations in an average case as well as the worst case because the height of the tree is limited to O(log n).

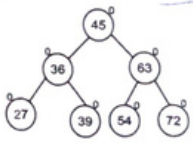


Figure 4.21 (a) left-heavy AVL Tree **(b)** right-heavy tree

The structure of AVL tree is much similar to binary search tree but with a little difference i.e. *balance factor*.

Balance factor = Height (left sub-tree)- Height (right sub-tree)

- If balance factor of a node is 1, then it means that the left subtree of the tree is one level higher than that of the rightsub-tree.
- If the balance factor of a node is 0 , then it means that the height of the left sub-tree is equal to the height of the right sub-tree.
- If the balance factor is -1, then it means that the left-sub tree of the tree is one level lower than that of the right sub-tree.

In above figure , Note that the nodes 18,39,54,and 72 have no children ,so their balance factor=0. Node 27 has one left child and zero right child. so, the height of left sub tree =1 whereas the height of the right sub-

tree =0. Thus, its balance factor=1. Look at node 36 , it has a leftsub-tree with height =2, whereeas the height of right sub-tree =1. Thus ,its balance factor =2-1=1 . Similiarly, the balance factor of node 45=3-2=1; node 63 has abalance factor 0 (1-1).

4.6 M-way Search Tree

As we discussed that binary search tree has a value and two pointer left and right , which point to the node's left and right sub-trees repectively.Same concept with multi-way search tree(M-way tree) which has M-1 values per node and M sub trees. In such a tree M is called the degree of the tree. In binary search tree degree is always 2, so have one value and two sub-trees. So we can say, in M-way tree every internal node consists of pointers to M sub-trees and contain M-1 keys, where M>2.

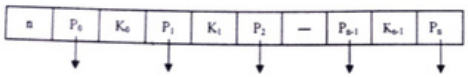


Figure 4.22

In the structure of an M-way search tree, shown in fig.4.22 $P_0P_1...P_n$ are pointers to the node's sub-trees and K_1K_2 K_{n-1} are the key values of the node. All the key values are stored in ascending order. That is, $K_i < K_{i+1}$ for $0 \leq i \leq n-2$.

In an M-way search tree , it is not compulsory that every node has exactly M-1 values and M sub-trees. Rather, the node can have anywhere from 1 to M-1 values, and the number of sub-trees can vary from 0(for a leaf node) to i+1, where i is the number of key values in the node . M is thus a fixed upper limit that defins how many key values can be stored in the node. In given figure 4.23 M-way search tree of order, M=3, so a node can store a maximum of two key values and can contain pointers to three sub-trees.

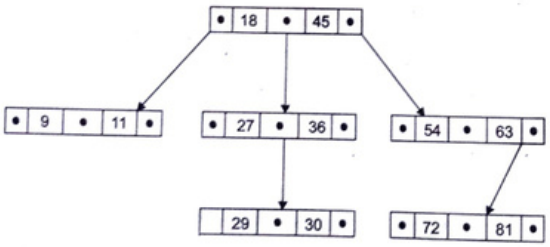


Figure 4.23 M-way search tree of order 3

Properties of M-way search tree:

1. Key values in the sub-pointed by P_0 are less than the key value K_0 . Similiarly, all the key values in the sub-tree pointed by P_1 are less than K_1, so on and so forth.So , generlised rule is that all the key values in the sub-tree pointed by P_i are less than K_i , where $0 \leq i \leq n-1$.
2. Key value in the sub-tree pointed by P_1 are greater than the key value K_0 Similarly, all the key values in the sub-tree pointed by P_2 are greater than K_2 , so on forth. So, generlised rule is that all the key values in the sub-tree pointed by P_i are greater than K_{i-1} , where $0 \leq i \leq n-1$
3. In an M-way search tree, every sub-tree is also an M-way search tree and follow the same rules.

4.7 Heap Tree

A complete binary tree whose every node satisfy a property

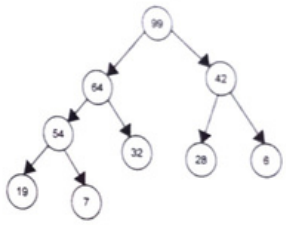


Figure 4.24 Max-heap

IF B is a child of A, then key(A)≥ key(B)

This property shows that elements at every node will be either greater than or equal to the element at its left and right child.Such a tree called as Binary Heap.

The root node has the heighest key value in the heap.Such heap commonly known as *max-heap*.

Similarly,elements at every node will be less than or equal to the element at its left and right child. Thus the root has the lowest key value.Such heap is called as *min-heap*.

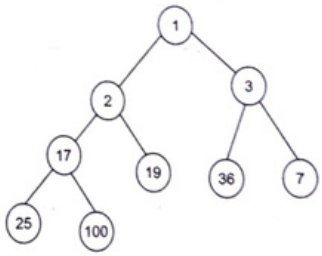


Figure 4.25 Min-heap

A binary heap is useful data structure in which elements can be added randomly but only the element with heighest value is removed in case of max heap and lowest value in case of min heap. A binary tree is an efficient data structure , but a binary heap is more space efficient and simpler.

4.8 Hauffman Trees

Entropy encoding is a type of lossless data compression technique to compress digital data by representing frequently occurring patterns with few bits and rarely occurring patterns with many bits. Hauffman coding is a type of entropy encoding. Hauffman coding algorithm is developed by David A.Hauffman. The key idea

behind Hauffman algorithm is that it encodes the most common characters using shorter strings of bits than those used for less common source characters.

The algorithm works by creating a binary tree of nodes that are stored in array. A node can be either a leaf node or an internal node .

Technique: Given n nodes and their weights, the Hauffman algorithm is used to find a tree with a minimum –weighted path length. The tree begins by creating a new node whose children are two nodes with the smallest weight, such that the new node's weight is equal to the sum of children's weight . In such way, two nodes are merged into one node. This process is going on until the tree has only one node.Such a tree with only one node is known as Hauffman tree.

Hauffman algo:-

Step 1: create a leaf node for each character. Add the character and its weight or prequency of occurence to the priority queue.

Step 2:Repeat step 3 to 5 while the total number of nodes in the queue is greater then 1.

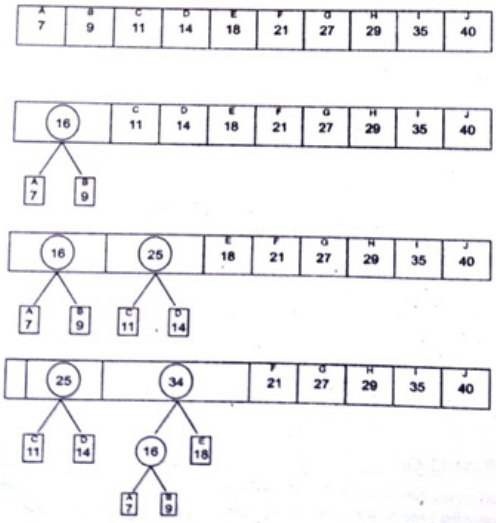
Step 3: Remove two nodes that have the lowest weight (or hight priority)

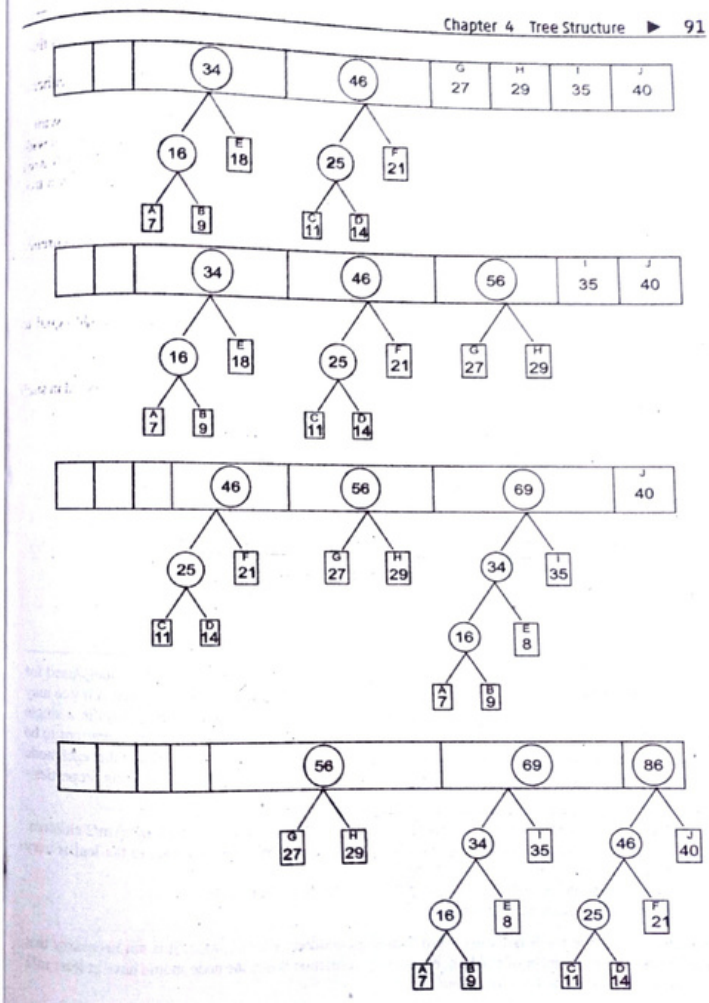
Step 4: Create a new internal node by merging these two nodes as chelidren and with weight equal to sum of the two node's weight.

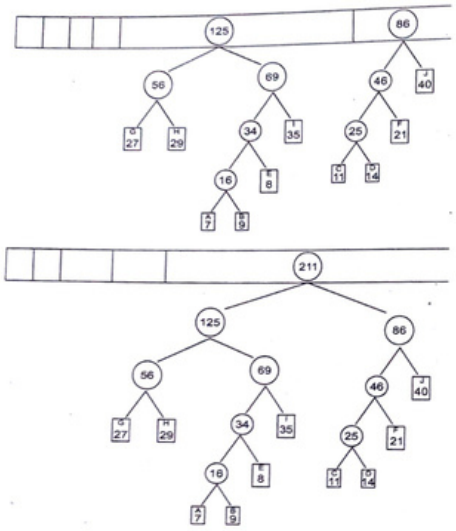
Step 5: Add the newly created node to the queue.

The Hauffman algorithm can be implemented using priority queue in which all the nodes are placed in such a way that the node with the lowest weight is given heighesy priority

Example: Create a Huffman tree with the following nodes arranged in priority Queue.







4.9 B Tree

B tree is a specialized m-way tree developed by Rudolf Bayer and Ed McCreight in 1970 , widely used for disk access. B tree of order m can have a maximum of m-1 keys and m pointers to its sub-trees. A b tree may contain a large number of key values and pointers to sub-trees, storing a large number of keys in a single nodes keeps the height of the tree relatively small. A B tree allows search, insertion, deletion operations to be performed in logarithmic amortized time. B tree of order m (the maximum number of children that each node can have) is a tree with all its properties of an M-way search tree. In addition it has the following properties:-

1. Every node in the B tree has at most (maximum) m children.
2. Every node in the b tree except the root node and the leaf node has at least (minimum) m/2 children. This condition helps to keep the tree bushy so that the path from the root node to the leaf is very short, even in a tree that stores of a lot of data.
3. The root node has at least two children if it is not a terminal (leaf) node.
4. All the leaf nodes are at the same level.

All internal nodes in the B tree can have n number of children, where $0 \le n \le m$. It is not necessary that every node has the same number of children, but the only restriction is that the node should have at least m/2 children. As a B tree of order 4 is given as follow:-

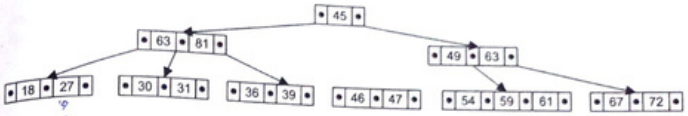


Figure 4.26 B Tree

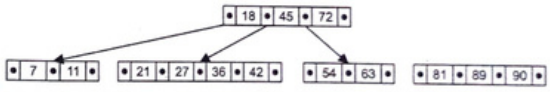
Operations like searching, insertion, deletion of an element from a B tree can be done similar to that in Binary search tree. Running time of the search operation depends upon the height of the tree, the algorithm to search an element in a B tree takes O(log n) time to execute.

Inserting a New Element in a B tree

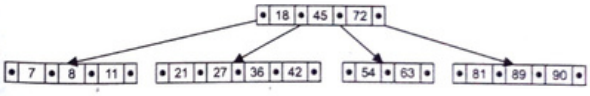
1. Search a B tree to find the leaf node where the new key value should be inserted.
2. If the leaf node is not null, that is, it contain less than m-1 key values, then insert the new element in the node keeping the node's elements ordered.
3. If the leaf node is full, that is, the leaf node already contains m-1 key values, then
   (a) Insert the new value in order into the existing set of keys,
   (b) Split the node at its median into two nodes (note that the split nodes are half full), and
   (c) Push the median element up to its parent's node .If parent's node is already full, then split the parent node by following same steps.

Example: Inserting a new element in B tree.

Look at the B tree of order 5 diven below and insert 8,9.



Step 1: Insert 8



Step 2: Insert 9

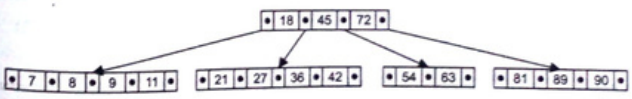


Figure 4.27 Insert 8,9 in B tree

Deleting an Element from a B tree

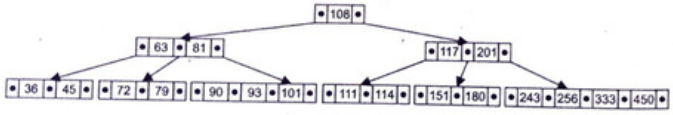
Like insertion, deletion is also done from the leaf nodes. There are two cases of deletion. In the first case a leaf node has to be deleted. In the second case an internal node has to be deleted. Let us first see the steps involved in deleting a leaf node.

1. Locate the leaf node which has to be deleted.
2. If the leaf node contains more than the minimum number of key values (more than m/2 elements), then delete the value.
3. Else if the leaf node does not contain m/2 elements, then fill the node by taking an element either from the left or from the right siblings.
   (a) If the left sibling has more than minimum number of key values, push its largest key into it's parent's node and pull down the intervening element from the parent node to the leaf node where the key is deleted.
   (b) Else, if the right siblings has more than the minimum number of key values , push its smallest key into its parent node and pull down the intervening element from the parent node to the leaf node where the key is deleted.
4. Else, if both the lefty and right siblings contain only the minimum number of elements , then create a new leaf node by (ensuring that the number of elements does not exceed the maximum number of elements a node can have, that is, m). If pulling the intervening element from the parent node leaves it with less than the minimum number of keys in the node, then propagate the process upwards, thereby reducing the height of the B tree.

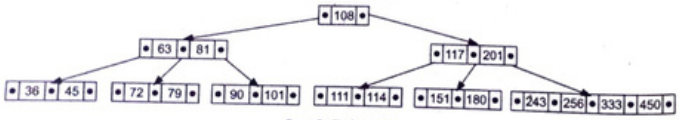
To delete an internal nodes, promote the successor or predecessor of the key to be deleted to occupy the position of the deleted key. This predecessor or successor will always be in the leaf node. So the processing will be done as if a value from the leaf node has been deleted.

Example: Deleting an element from B tree.

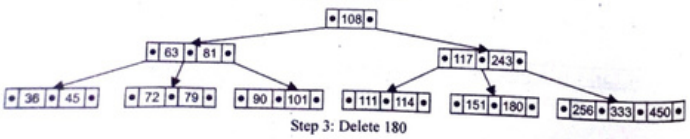
Consider the following B tree of order 5 and delete values 93,201 and 180.



Step 1: Delete 93



Step 2: Delete 201



Step 3: Delete 180

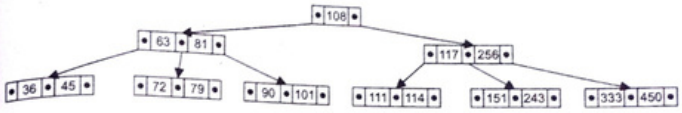


Figure 4.28 Delete 93, 201, 93 from B tree

4.10 B+ Tree

An efficient insertion, retrieval, and removal of records, can be done by using B+ tree (a variant of a B tree). B+ tree stores sorted data , each of which is identified by a key. While B tree stores both keys and data records at interior nodes, a B+ tree, in contrast , stores all the records at the leaf level of the tree; only keys are stored in their interior nodes.

The leaf nodes of a B+ tree are often linked to another in a linked list. This has added advantage of making the queries simpler and more efficient.

B+ tree are used to store large amount of data that cannot be stored in the main memory . With B+ tree, the secondary storage(magnetic disk) is used to store the leaf nodes of trees and internal nodes of trees are stored in the main memory.

B+ trees store data only in the leaf nodes. All other nodes(internal nodes) are called index nodes or i-nodes and store index values. This allow us to traverse the tree from the root down to the leaf node that stores the desired data item.Fig. shows a B+ tree of order 3.

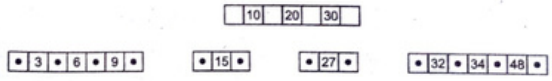
Insertion a New Element in a B+ tree

A new element is simply added in the leaf node if there is space for it. But if a data node in the tree where insertion has to be done is full, then that node is split into two nodes. This calls for adding a new index value in the parent index node so that future queries can arbitrate between the two new nodes.

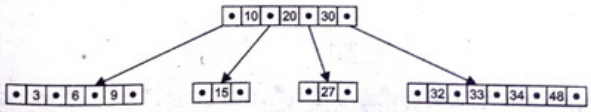
However , adding the new index value in the parent node may cause it,turn ,to spilt . In fact, all the nodes on the path from a leaf to the root may be spilt when a new value is added to a leaf node. If the root node splits, a new leaf node is created and the tree grows by one level.

Example: Insert a new element in B+ tree.

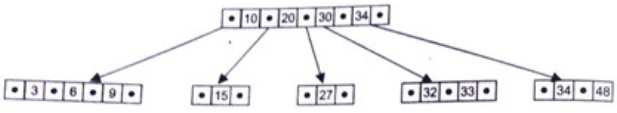
Consider the B+ tree of order 4 given and insert 33 in it.



Step 1: Insert 33



Step 2: Split the leaf node



Step 3: Split the index node

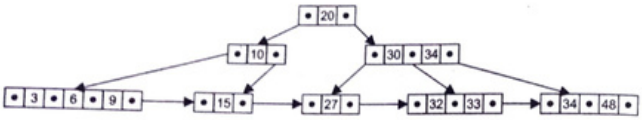


Figure 4.29 Inserting node 33 in the B+ tree

Deleting an Element from a B+ tree

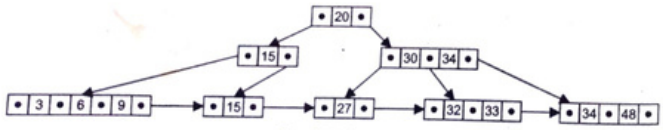
As in B tree, deletion is always done from a leaf node. If deleting a data element leaves that node empty, then the neighbouring nodes are examined and merged with the underfall node.

This process calls for the deletion of an index value from the parent index node which, in turn, may cause it to become empty. Similar to the insertion process, deletion may cause a merge-delete wave to run from a leaf node all the way up to root.This leads to shrinking of the tree by one level.

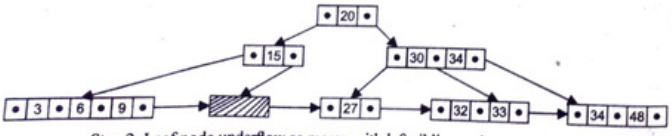
Many database system are implemented using B+ tree structure because of its simplicity. Since all the data appear in the leaf nodes and are ordered , the tree is always balanced and makes searching for data efficient. A B+ tree can be thought as a multi-level index in which the leaves make up a dense index and the non-leaf nodes make up a sparse index.

Example: Deleting a element from B+ tree.

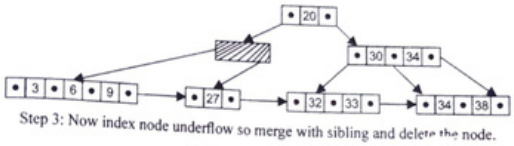
Consider the B+ tree of order 4 given below and delete node 15 from it.



Step 1: Delete 15



Step 2: Leaf node underflow so merge with left sibling and remove key 15



Step 3: Now index node underflow so merge with sibling and delete the node.

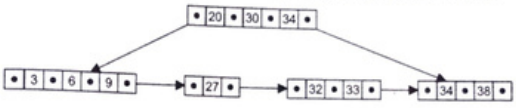


Figure 4.30 Deleting node 15 from the given B+ tree

Comparisions between B trees and B+ trees

B tree	B+ tree
1 Search keys are not repeated	1 Stores redundant search keys
2 Data is stored in internal or leaf nodes	2 Data is stored only in leaf nodes.
3 Searching takes more time as data may be found in a leaf or non-leaf node.	3 Searching data is easy as the data can be found in leaf nodes only.
4 Deletion of non-leaf nodes is very complicated	4 Deletion is very simple because data will be in the leaf nodes
5 Leaf nodes cannot be stored using linked list	5 Leaf nodes data are ordered using sequential linked lists
6 The structure and operations are complicated.	6 The structure and operations are simple.

■ ■ ■

Very Short Questions

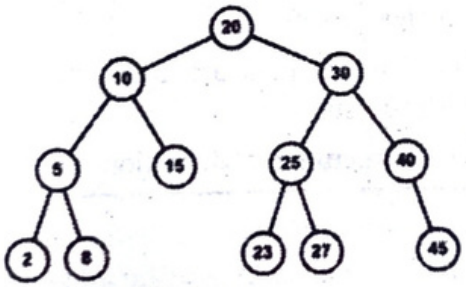
1. Define tree.
2. What is complete binary tree?
3. What is full binary tree?
4. What is extended binary tree?
5. What do you understand by Binary search tree?
6. What do you understand by traversal of tree?
7. State the list of operation which can be performed on Binary search tree?
8. What is AVL tree?
9. Draw a binary expression tree that represents the following postfix expression
   AB+C*D
10. What is the maximum number of nodes that can be found in a binary tree at level 3,4 and 12?

Short Questions

1. Write short notes on
   a. Tournament tree b. Expression tree
   b. Forests d. General tree
2. Convert the prefix expression -/ab*+bcd into infix expression and then draw the corresponding expression tree.
3. What are two ways of representating binary trees in memory ? Which one do you prefer and why?
4. Write a Short note on
   a) M-way tree b). B tree
5. Discuss the run time complexity of Binary search tree.

Long Questions

1. Explain the concept and operations performed on Binary search tree.
2. Explain AVL tree and how AVL tree is better than a binary search tree.
3. Create a binary search tree with input given below
   98,2,48,12,56,32,4,67,23,87,23,55.46
   (a) Insert 21,39,45,54,and 63 into the trees
   (b) Delete values 23,56,2, and 45 from the tree
4. Consider the binary search tree given below
   Now do the following operation :
   a) Find the result of in-order, pre-order and post-order traversals.
   b) Insert 11,22,33,and 44



5. **Explain the concept of Hauffman's tree.**