APPLICATIONs OF TREES

- Trees are used to store simple as well as complex data. Here simple means an integer value, character value and complex data means a structure or a record.
- Trees are often used for implementing other types of data structures like hash tables, sets, and maps.
- A self-balancing tree, Red-black tree is used in kernel scheduling, to preempt massively multiprocessor computer operating system use.
- Another variation of tree, B-trees are prominently used to store tree structures on disc. They are used to index a large number of records.
- B-trees are also used for secondary indexes in databases, where the index facilitates a select operation to answer some range criteria.
- Trees are an important data structure used for compiler construction.
- Trees are also used in database design.
- Trees are used in file system directories.
- Trees are also widely used for information storage and retrieval in symbol tables.

www.binils.com

BINARY SEARCH TREES

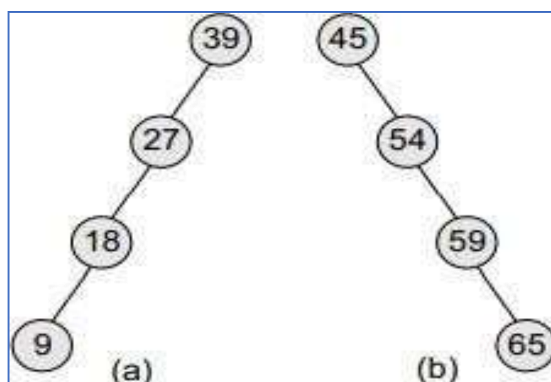
A binary search tree, also known as an ordered binary tree, is a variant of binary trees in which the nodes are arranged in an order. In a binary search tree, all the nodes in the left sub-tree have a value less than that of the root node. Correspondingly, all the nodes in the right sub-tree have a value either equal to or greater than the root node. The same rule is applicable to every sub-tree in the tree.

A binary search tree is a binary tree with the following properties:

- The left sub-tree of a node N contains values that are less than N's value.
- The right sub-tree of a node N contains values that are greater than N's value.
- Both the left and the right binary trees also satisfy these properties and, thus, are binary search trees.

Download Binils Android App in Playstore Download Photoplex App

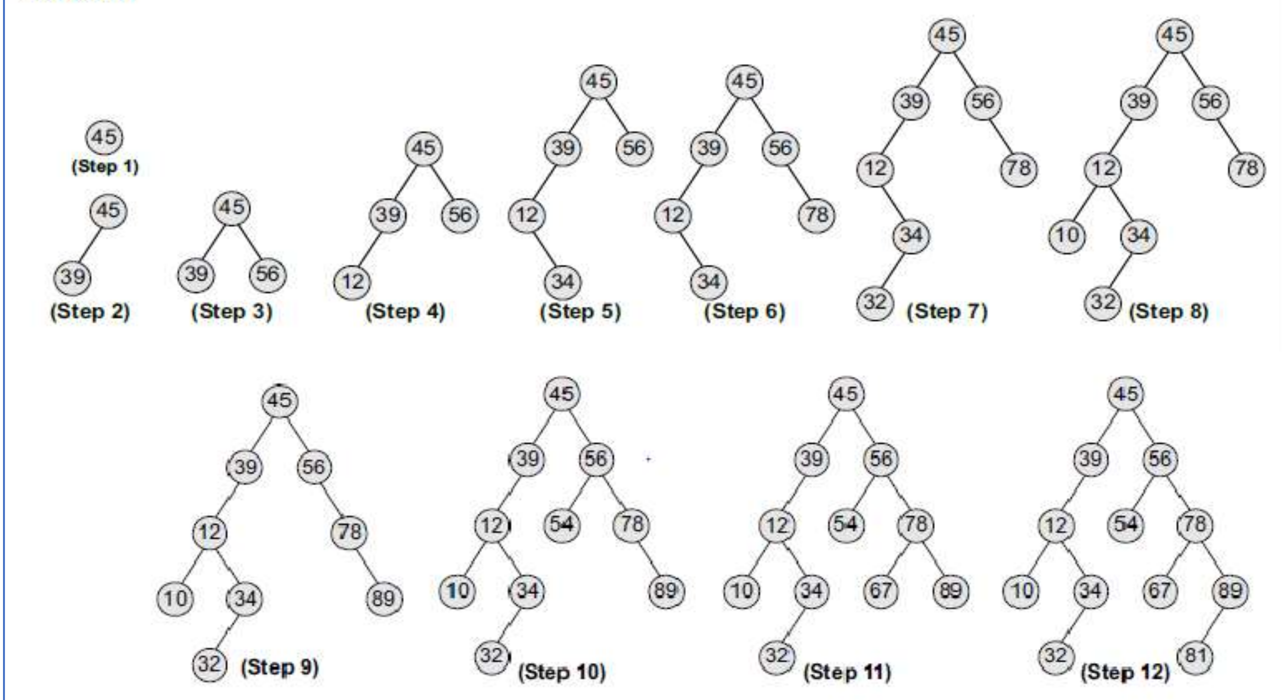
Binary search trees are considered to be efficient data structures especially when compared with sorted linear arrays and linked lists. In a sorted array, searching can be done in O(logn) time, but insertions and deletions are quite expensive. In contrast, inserting and deleting elements in a linked list is easier, but searching for an element is done in O(n) time. However, in the worst case, a binary search tree will take O(n) time to search for an element. The worst case would occur when the tree is a linear chain of nodes as given in Fig.



(a) Left skewed, and (b) right skewed binary search trees Example 1

www.binils.com

Create a binary search tree using the following data elements: 45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67, 81



OPERATIONS ON BINARY SEARCH TREES

The operations on binary search tree are

1. Search
2. Insert
3. Delete
4. Determine the number of node
5. Determine the height
6. Fining largest and smallest element in BST

1. Searching for a Node in a Binary Search Tree

The search function is used to find whether a given value is present in the tree or not. The searching process begins at the root node. The function first checks if the binary search tree is empty. If it is empty, then the value we are searching for is not present in the tree.

www.binils.com

- However, If there are nodes in the tree, then the search function checks to see if the key value of the current node is equal to the value to be searched.
- If not, it checks if the value to be searched for is less than the value of the current node, in which ase it should be recursively called on the left child node.
- In case the value is greater than the value of the current node, it should be recursively called on the right child node.

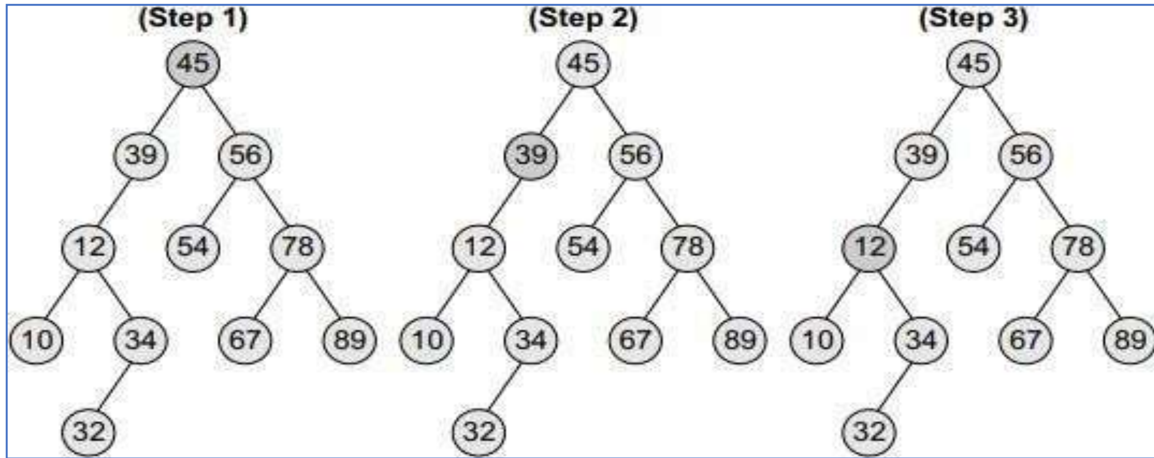
```
searchElement (TREE, VAL)

Step 1: 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
```

Algorithm to search for a given value in a binary search tree

Example 1:

Searching a node with value 12 in the given binary

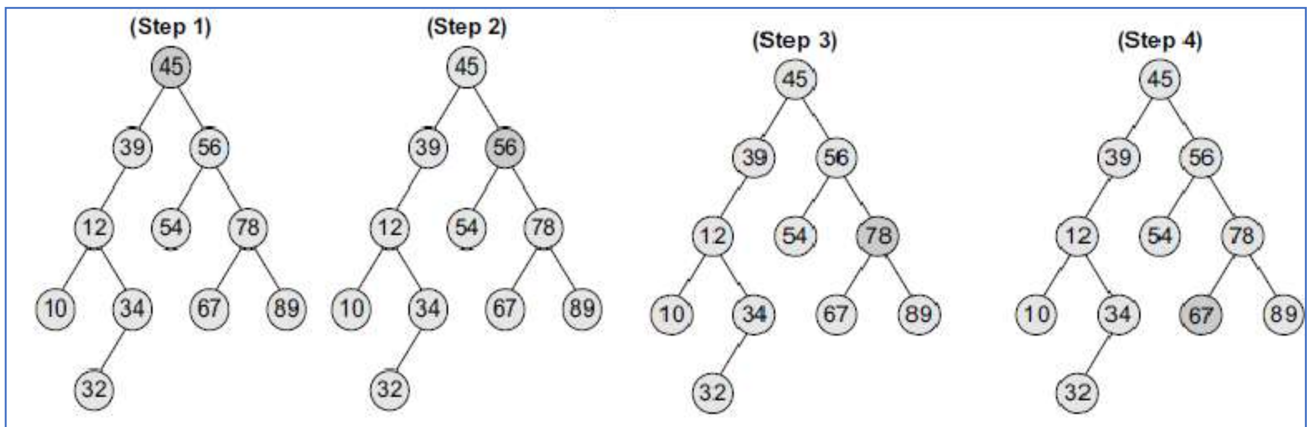
search tree Solution :



www.binils.com

Example 2: Searching a node with value 67 in the given binary

search tree Solution :

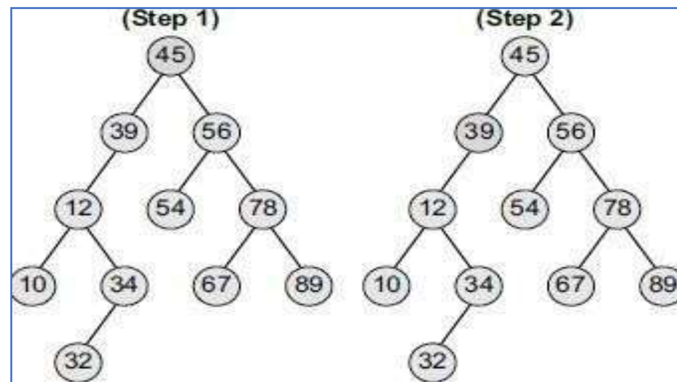


Example 3: Searching a node with the value 40 in the given binary

search tree Solution :

The procedure to find the node with value 40 is shown in Fig. The search would terminate after reaching

node 39 as it does not have any right child.



2. Insertion

- The insert function is used to add a new node with a given value at the correct position in the binary search tree. Adding the node at the correct position means that the new node should not violate the properties of the binary search tree. The insert function is similar to the search function.
- This is because we first find the correct position where the insertion has to be done and then add the node at that position. 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.

www.binils.com

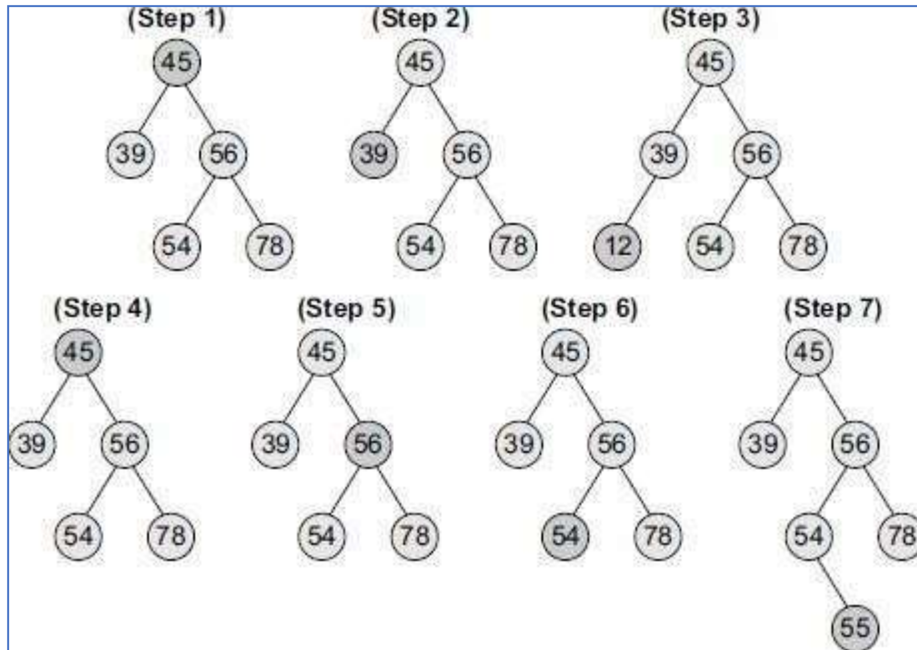
```
Insert (TREE, VAL)

Step 1: 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]
Step 2: END
```

Algorithm to insert a given value in a binary search tree

The insert function requires 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.

Download Binils Android App in Playstore Download Photoplex App

Example 1: Inserting nodes with values 12 and 55 in the given binary search tree



3. Deleting a Node from a Binary Search Tree

www.binils.com

The delete function deletes a node from the binary search tree. However, utmost care should be taken that the properties of the binary search tree are not violated and nodes are not lost in the process. We will take up three cases in this section and discuss how a node is deleted from a binary search tree.

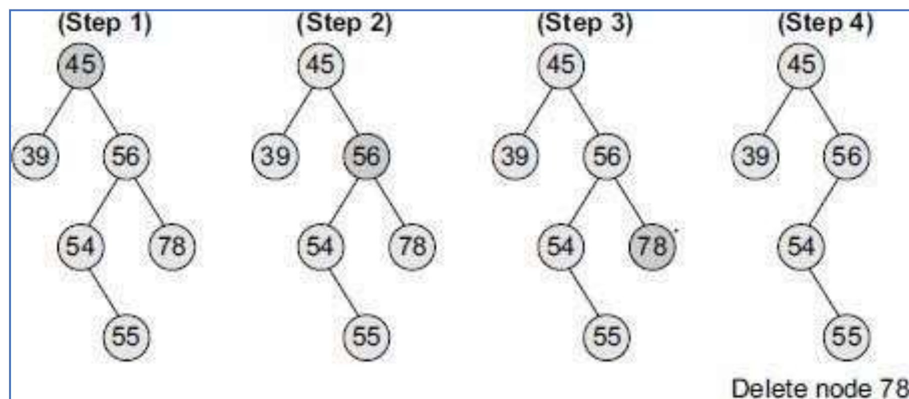
```
Delete (TREE, VAL)

Step 1: 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
```

Algorithm to delete a node from a binary

search tree Case 1: Deleting a Node that has

No Children

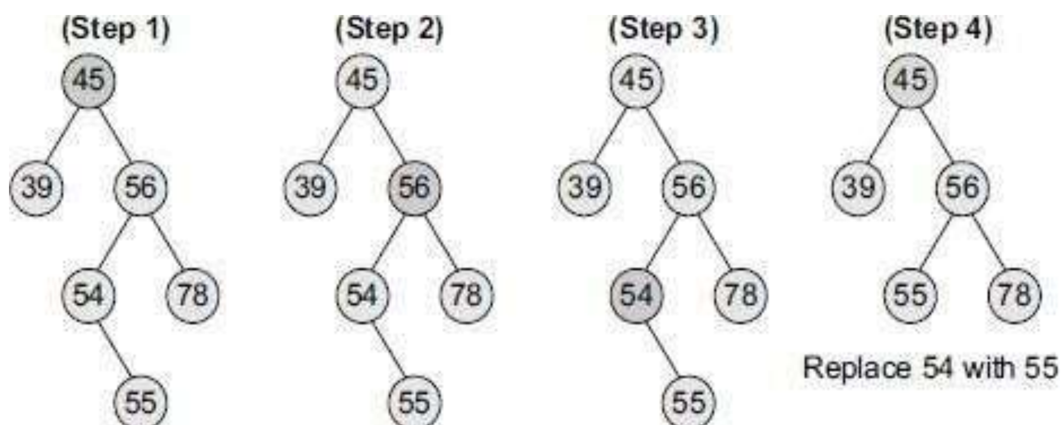
If we have to delete node 78, we can simply remove this node without any issue. This is the simplest case of deletion



Case 2: Deleting a Node with One Child

www.binils.com

To handle this case, the node's child is set as the child of the node's parent. In other words, replace the node with its child. Now, if the node is the 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. Look at the binary search tree shown in Fig. and see how deletion of node 54 is handled.

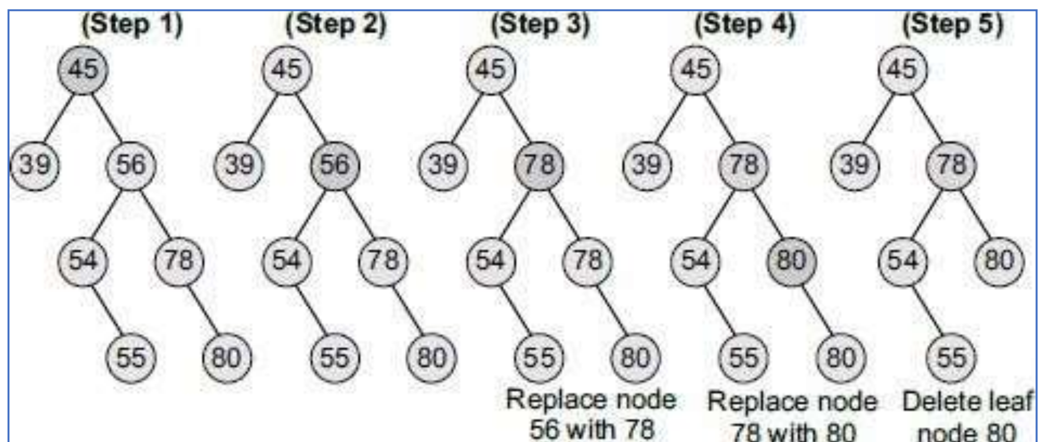


Case 3: Deleting a Node with Two Children

To handle this case, replace the node's value with its in-order predecessor (largest value in the left sub-tree) or in- order successor (smallest value in the right sub- tree). The in-order predecessor or the successor can then be

Download Binils Android App in Playstore Download Photoplex App

deleted using any of the above cases. Look at the binary search tree given in Fig. and see how deletion of node with value 56 is handled. This deletion could also be handled by replacing node 56 with its in-order successor.



4. Determining the Height of a Binary Search Tree

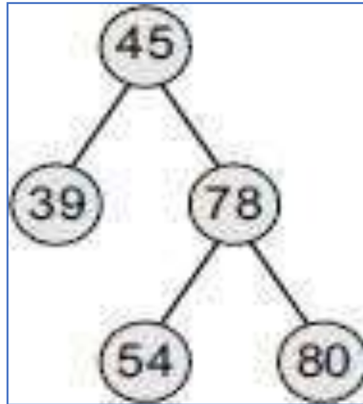
In order to determine the height of a binary search tree, we calculate the height of the left sub-tree and the right sub-tree. Whichever height is greater, 1 is added to it.

www.binils.com

Algorithm to determine the height of a binary search tree

```
Height (TREE)

Step 1: 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]
Step 2: END
```

For example, if the height of the left sub-tree is greater than that of the right sub-tree, then 1 is added to the left sub- tree, else 1 is added to the right sub-tree. Look at given Fig . Since the height of the right sub-tree is greater than the height of the left sub-tree, the height of the tree = height (right sub-tree) + 1= 3



Binary search tree with height = 3

Deleting a Binary Search Tree

To delete/remove an entire binary search tree from the memory, we first delete the elements/nodes in the left sub- tree and then delete the nodes in the right sub-tree. The algorithm shown in below Fig. gives a recursive procedure to remove the binary search tree.

Algorithm to delete a binary

www.binils.com

```
deleteTree(TREE)

Step 1: IF TREE != NULL
            deleteTree (TREE -> LEFT)
            deleteTree (TREE -> RIGHT)
            Free (TREE)
        [END OF IF]
Step 2: END
```

AVL TREES

- AVL tree is a self-balancing binary search tree invented by G.M. Adelson-Velsky and E.M. Landis in 1962.

  In an AVL tree, the heights of the two sub-trees of a node maydiffer by at mostone. Duetothisproperty, the ***AVL tree*** isalso known as a ***height-balanced tree.***

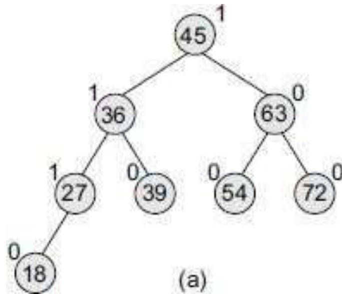
- The key advantage of using an AVL tree is that it takes ***O(log n)*** time to perform search, insert, and deleteoperations inanaverage caseaswellas theworst case because the height of the tree is limited to ***O(log n).***
- The structure of an AVL tree is the same as that of a binary search tree but with a little difference. In its structure, it stores an additional variable called the ***Balance Factor.***
- The balance factor of a node is calculated by subtracting the height of its right sub-tree from the height of its left sub-tree. A binary search tree in which every node hasabalance factor of ***–1, 0, or 1*** is saidto be height balanced. A node with any other balance factor is considered to be unbalanced and requires rebalancing of the tree.

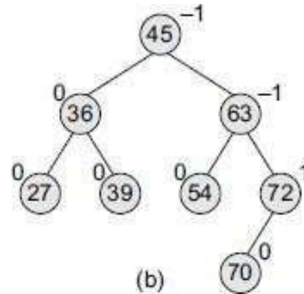
Balance factor = Height (left sub-tree) – Height (right sub-tree)

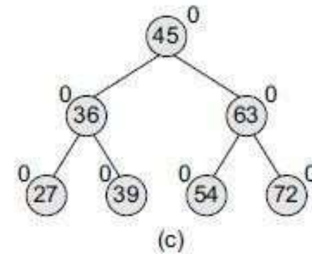
- If the balance factor of a node is 1, then it means that the left sub-tree of the tree isonelevel higherthan thatofthe right sub-tree. Such a tree is therefore calledas a ***left-heavy tree.***
- If the balance factor ofanode is 0, then it meansthat theheight ofthe left sub-tree(longestpathintheleft sub-tree)isequaltotheheightoftheright sub-tree.
- If the balance factor ofa node is –1, then it means that the left sub-treeofthe tree is one level lowerthan thatoftherightsub-tree.Suchatreeisthereforecalledas a ***right-heavy tree.***

An AVL tree is a balanced binary search tree. In an AVL tree, balance factor of every node

The trees given in given Fig. aretypicalcandidates of AVL trees because the balancing factor ofevery node is either 1, 0, or –1. However, insertions and deletions froman AVL tree may disturb the balance factor ofthe nodesand,thus, rebalancingofthetreemay have to bedone.



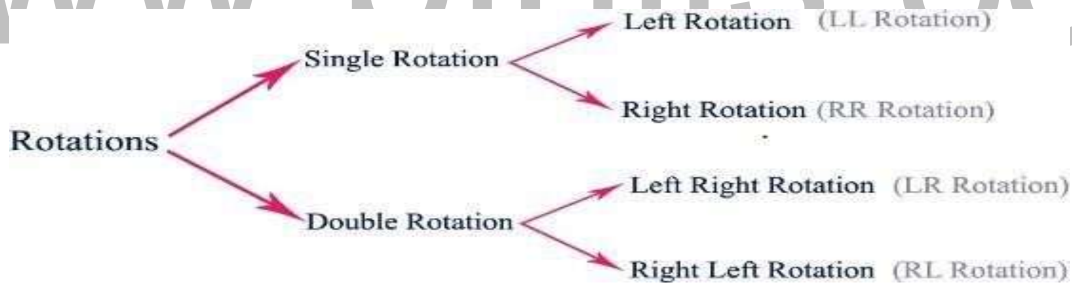




(a) Left-heavy AVL tree **(b) Right-heavy tree** **(c) Balanced tree**

Thetreeisrebalanced byperforming ***rotation*** atthecriticalnode.Therearefour ***types of rotations***:

www.binils.com



OPERATIONS on AVL TREE

1. **Search**
2. **Insert**
3. **Delete**

1. Searching for a Node in an AVL Tree

Searching in an AVL tree is performed exactly the same way as it is performed in a binary search tree. Due to the height-balancing of the tree, the search operation takes **O(log n**) time to complete. Since the operation does not modify the structure of the tree, no special provisions are required.

Step 1: Read the search element from the user

Step 2: Compare, the search element with the value of root node in the tree.

Step 3: If both are matching, then display "Given node found!!!" and terminate the function

Step 4: If botharenotmatching,then checkwhethersearchelement issmalleror larger than that node value.

www.binils.com

Step 5: If search element is smaller, then continue the search process in left subtree.

Step 6: If search element is larger, then continue the search process in right subtree.

Step 7: Repeat the same until we found exact element or we completed with a leaf node **Step 8:** Ifwereachto thenodewithsearchvalue,thendisplay"Element isfound" and terminate thefunction.

Step 9: If we reach to a leaf node and it is also not matching, then display "Element not found" and terminate the function.

2. Inserting a New Node in an AVL Tree

In an AVL tree, the insertion operation is performed with **O(log n)** time complexity. In AVL Tree, new node is always inserted as a leaf node. The insertion operation is performed as follows...

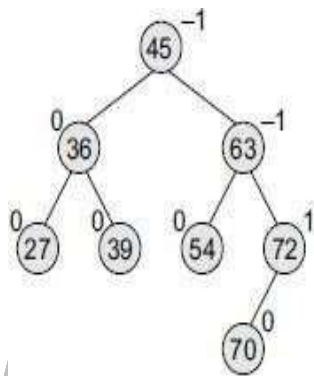
Step 1: Insert thenew element into thetree using Binary Search Tree insertion logic.

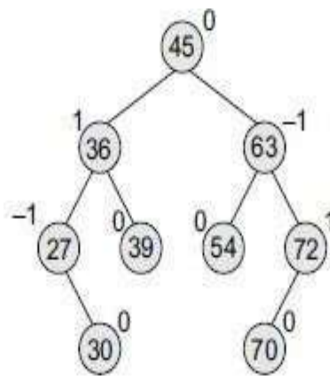
Step 2: After insertion, check the **Balance Factor** of every node.

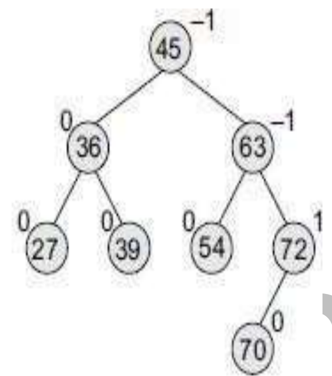
Step 3: If the **Balance Factor** of every node is **0 or 1 or -1** then go for next operation.

Step 4: If the **Balance Factor** of any node is other than **0 or 1 or -1** then tree is said to be imbalanced. Then perform the suitable **Rotation** to make it balanced. And go for next operation.

Consider the AVL tree given in Fig. If we insert a new node with the value 30, then the new tree will still be balanced and no rotationswillberequired inthis case andwhich shows the tree after inserting node 30.







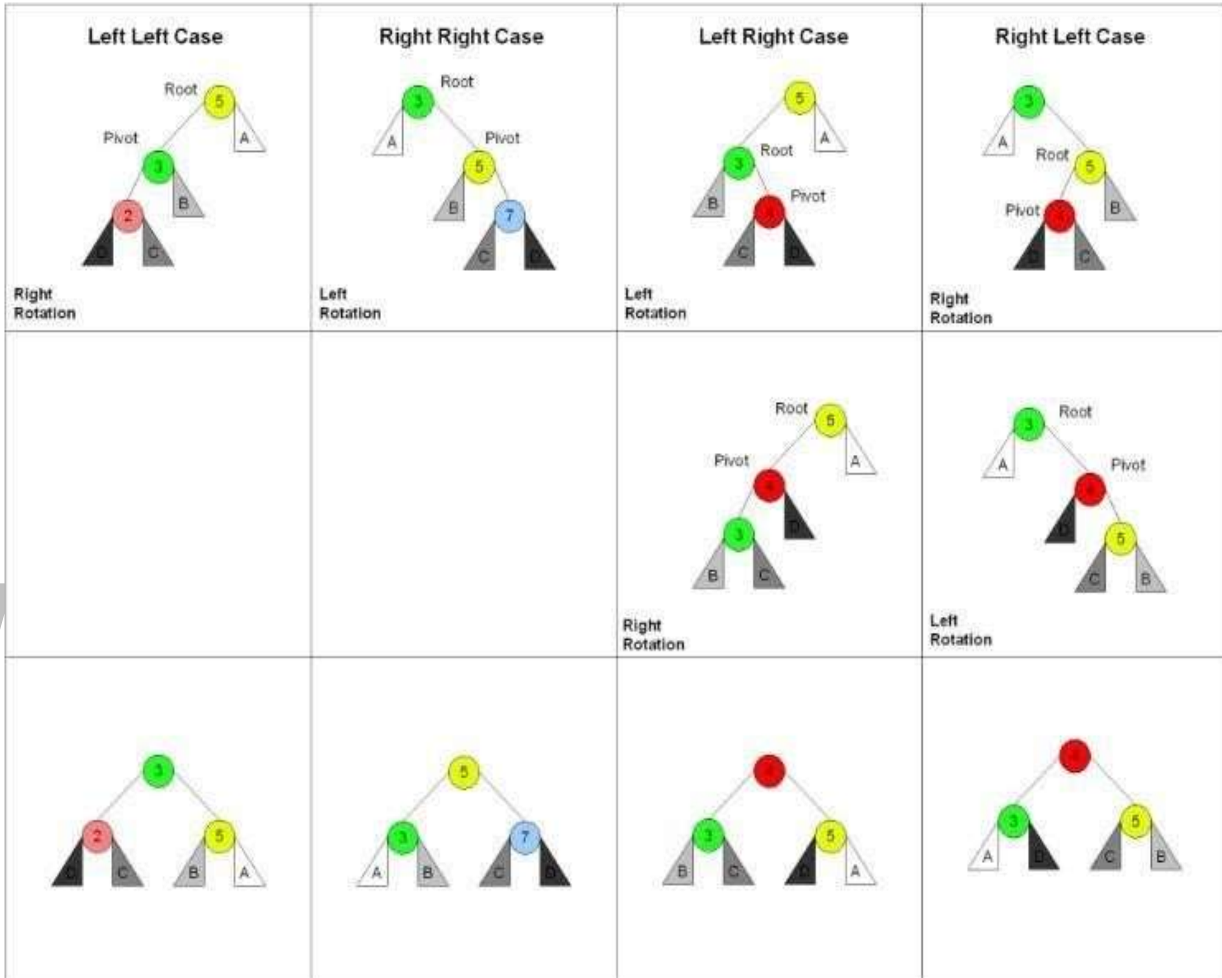
www.binils.com

AVL tree AVL tree after inserting AVLtreeanode with the value 30

The four categories of rotations are:

1. LLrotationThenewnode isinsertedintheleftsub-treeoftheleftsub-treeofthe critical node.
2. RRrotationThenewnodeis insertedintherightsub-treeoftherightsub-treeof the critical node.
3. LRrotation The new node is inserted inthe right sub-treeofthe left sub-treeof the critical node.
4. RLrotation The new node is inserted inthe left sub-treeoftheright sub-treeof the critical node.

LL rotation RR rotation LR rotation RL rotation



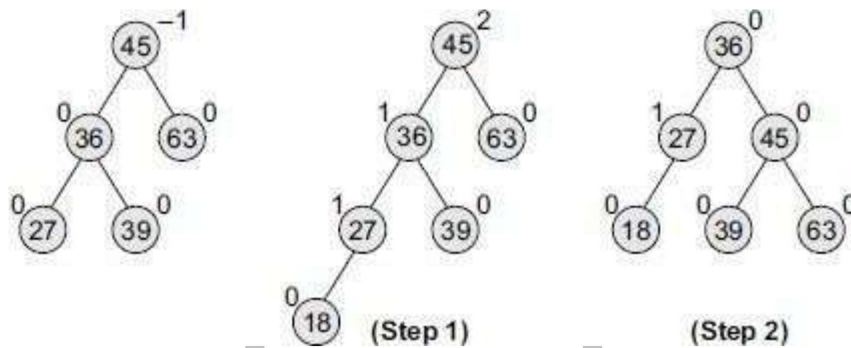
Download Binils Android App in Playstore Download Photoplex App

LL Rotation

Itisalsocalled asSingleleftrotation.ItRotates the edgeconnectingtherootand its right child in the binary tree

Example 1: ConsidertheAVLtreegiveninFig. andinsert 18intoit.

Solution:



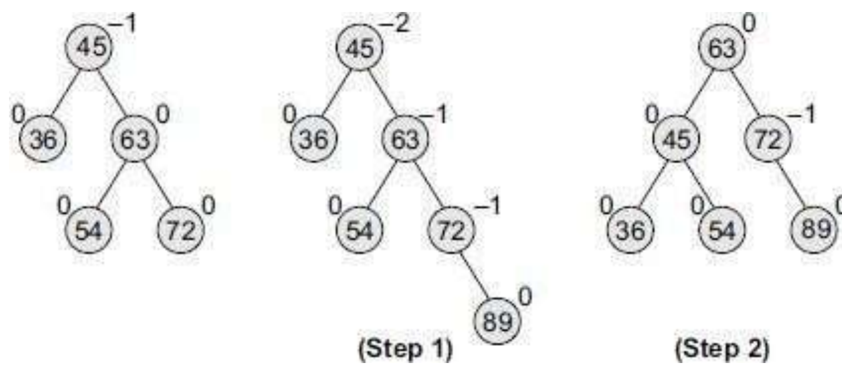
www.binils.com

RR Rotation

It is also called as Single right rotation. It Rotatesthe edge connecting the root and its left child in the binary tree

Example 2: Consider the AVL tree given in Fig. and insert 89 into it.

Solution:



LR Rotation

- It is also called as Double left-right rotation.
- Combination of tworotations

  1. perform left rotation of the left sub-tree of root r
  2. perform right rotation ofthe new tree rooted at r

- It is performed after a new key is inserted into the right sub-tree of the left child of a tree whose root had the balance of +1 before the insertion

RL Rotation

www.binils.com

- It is also called as Double right-leftrotation
- Combination of tworotations

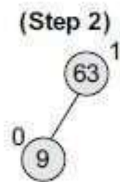
  1. perform right rotation ofthe right sub-tree of root r
  2. perform left rotation of the new tree rooted at r

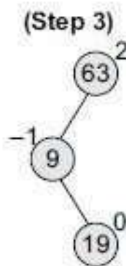
- It is performed after a new key is inserted into the left sub-tree of the right child of a tree whose root had the balance of -1 before the insertion

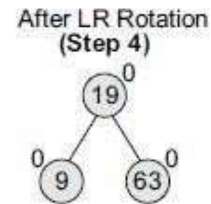
Example 1 ConstructanAVLtreebyinsertingthefollowingelementsinthegiven order. 63, 9, 19, 27, 18, 108, 99, 81.

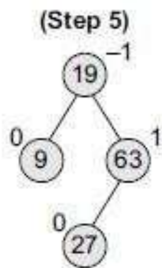
Download Binils Android App in Playstore Download Photoplex App

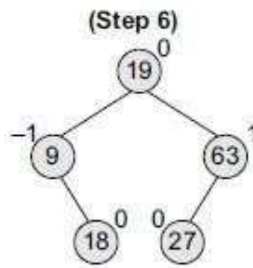
Solution:

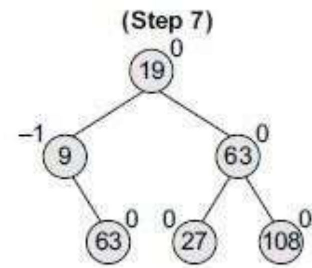


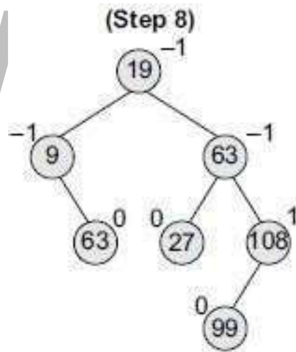


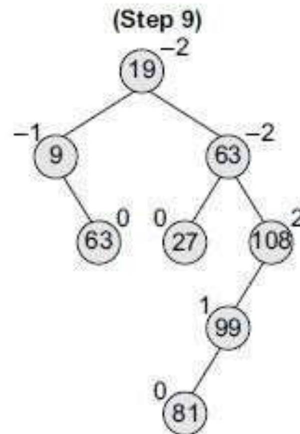


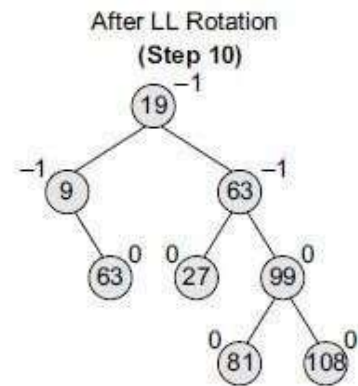










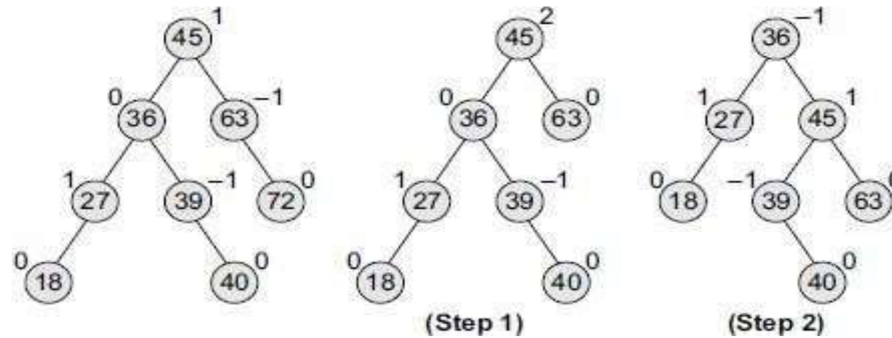


3. <u>Deleting a Node from an AVL Tree</u>

Deletion of a node in an AVL tree is similar to that of binary search trees. But, Deletionmay disturbthe AVL ness ofthetree, so to rebalancethe AVLtree, weneedto performrotations. Thereare two classes of rotations that can be performed on an AVL tree after deleting a given node. These rotationsare Rrotationand L rotation.

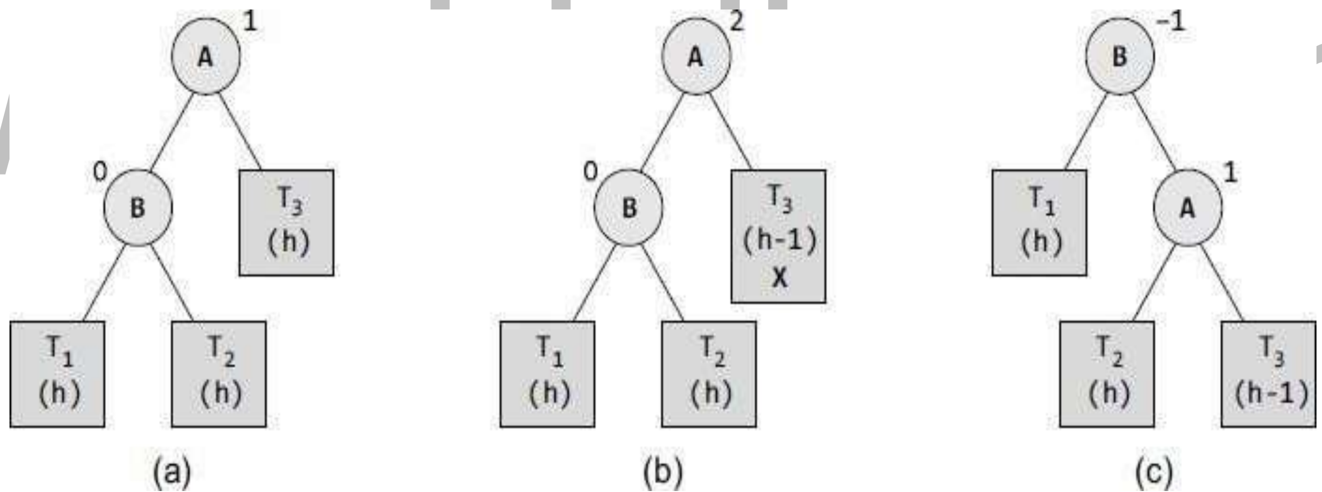
Example: ConsidertheAVLtreegiveninFig.anddelete72fromit.

Solution:



R0 Rotation

Let Bbethe root ofthe left or right sub-treeof A(critical node). R0 rotation is applied if the balance factor of B is 0.



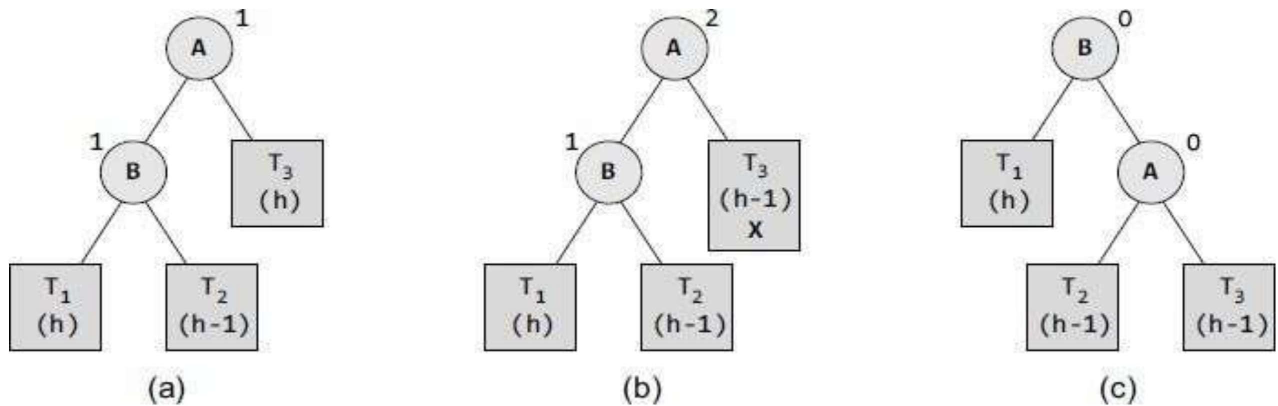
Tree (a) is an AVL tree. In tree (b), the node X is to be deleted from the right sub-tree of the critical node A(node A is thecritical node because it is the closest ancestor whose balance factor is not –1, 0, or 1). Since the balance factor of node B is 0, we apply R0 rotation as shown in tree (c). During the process of rotation, node B becomes the root, with T1 and A as its left and right child. T2 and T3 becometheleftandrightsub-treesof A.

R1 Rotation

Let B be the root of the left or right sub-tree of A (critical node). R1 rotation is applied if the balance factorofB is

1.Observethat R0 andR1 rotationsaresimilar to LLrotations; the only difference is that R0 and R1

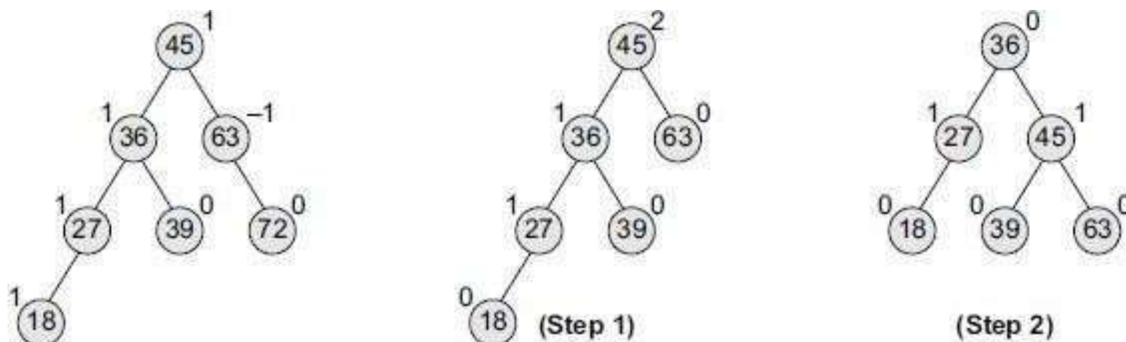
rotations yield different balance factors. Thisis illustrated in Fig.



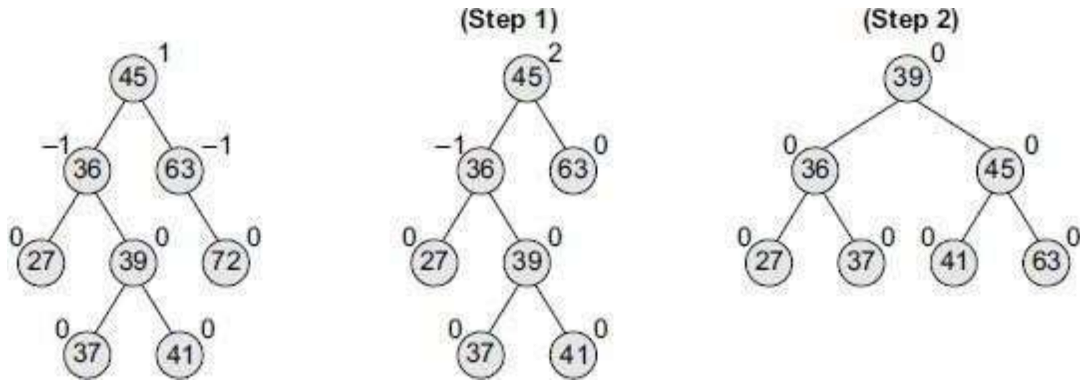
www.binils.com

Tree (a) is an AVL tree. In tree (b), the node X is to be deleted from the right sub-tree of the critical node A(node Aisthe criticalnode because it is theclosest ancestor whose balance factor is not –1, 0, or 1). Since the balance factor of node B is 1, we apply R1 rotation as shown in tree (c). During the process ofrotation, node B becomes the root, with T1 and A as its left and right children. T2 and T3 become the left and right sub- trees of A.

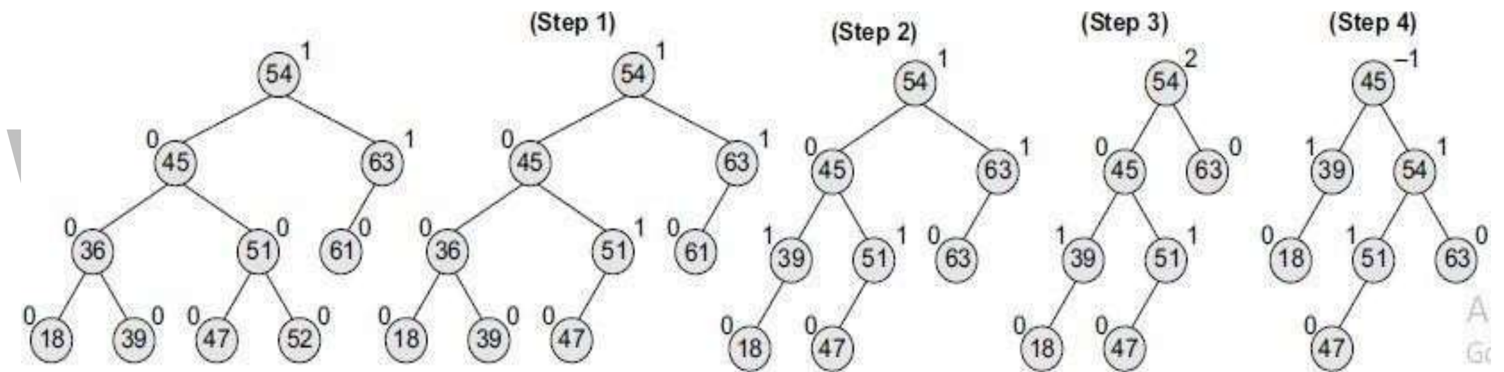
Example 1: Consider the AVL tree given in Fig. and delete 72 from it.



Example 2 Consider the AVL tree given in Fig. and delete 72 from it.



Example 3 Delete nodes 52, 36, and 61 from the AVL tree given in Fig.



Programming
Example #include
<stdio.h> #include
<stdlib.h>
// Create
Node struct
Node
{

```
        int key;
        struct Node *left;
        struct Node
        *right; int height;
};
int max(int a, int b);
// Calculate height
int height(struct Node *N)
{
  if (N == NULL)
    return 0;
  return N->height;
}
int max(int a, int b)
{
  return (a > b) ? a : b;
}
// Create a node
struct Node *newNode(int key)
```

www.binils.com

```
{
struct Node *node = (struct Node
 *) malloc(sizeof(struct Node));
 node->key = key;
 node->left =
 NULL; node-
 >right = NULL;
 node->height = 1;
 return (node);
}
// Right rotate
struct Node *rightRotate(struct Node *y)
 {
  struct Node *x = y->left;
  struct Node *T2 = x-
  >right; x->right = y;
  y->left = T2;
  y->height = max(height(y->left), height(y->right))
  + 1; x->height = max(height(x->left), height(x-
  >right)) + 1; return x;
```

www.binils.com

```
}
// Left rotate
struct Node *leftRotate(struct Node *x)
{
 struct  Node  *y  =  x-
 >right; struct Node *T2
 = y->left; y->left = x;
 x->right = T2;
 x->height = max(height(x->left), height(x->right))
 +  1;  y->height  =  max(height(y->left),  height(y-
 >right)) + 1; return y;
}
// Get the balance factor
int getBalance(struct Node *N)
{
 if (N == NULL)
   return 0;
 return height(N->left) - height(N->right);
}
```

www.binils.com

```
// Insert node

struct Node *insertNode(struct Node *node, int key)

{

  // Find the correct position to insertNode the node and

  insertNode it if (node == NULL)

          return

  (newNode(key)); if (key <

  node->key)

          node->left = insertNode(node->left,

  key); else if (key > node->key)

          node->right = insertNode(node->right, key);

  else

          return node;

  // Update the balance factor of each node and

// Balance the tree

  node->height = 1 + max(height(node-

  >left), height(node->right));

  int balance = getBalance(node);

  if (balance > 1 && key < node->left->key)
```

www.binils.com

```
 return rightRotate(node);


 if (balance < -1 && key > node->right-
  >key) return leftRotate(node);
 if (balance > 1 && key > node->left-
  >key) { node->left = leftRotate(node-
  >left); return rightRotate(node);
 }
 if (balance < -1 && key < node->right->key)
{
  node->right = rightRotate(node-
  >right); return leftRotate(node);
 }
 return node;
}


struct Node *minValueNode(struct Node *node)
{
 struct Node *current = node;
```

www.binils.com

```
  while (current->left !=
  NULL) current = current-
  >left; return current;
}
// Delete a nodes
struct Node *deleteNode(struct Node *root, int key)
{
  // Find the node and delete
  it if (root == NULL)
        return root;
  if (key < root-
  >key)
        root->left = deleteNode(root->left,
  key); else if (key > root->key)
        root->right = deleteNode(root->right, key);
  else
{
        if ((root->left == NULL) || (root->right == NULL))
        {
        struct Node *temp = root->left ? root->left : root->right;
```

www.binils.com

```
if (temp ==
NULL) { temp =
root;
 root = NULL;
}
else
 *root = *temp;
free(temp);
}
else {
  struct Node *temp = minValueNode(root-
  >right); root->key = temp->key;
  root->right = deleteNode(root->right, temp->key);
 }
}

if (root ==
 NULL) return
 root;
// Update the balance factor of each node and
```

www.binils.com

```
// balance the tree
 root->height = 1 + max(height(root->left), height(root-
 >right)); int balance = getBalance(root);
 if (balance > 1 && getBalance(root->left)
  >= 0) return rightRotate(root);
 if (balance > 1 && getBalance(root->left) <
  0) { root->left = leftRotate(root->left);
  return rightRotate(root);
 }
 if (balance < -1 && getBalance(root->right)
  <= 0) return leftRotate(root);
 if (balance < -1 && getBalance(root->right) >
  0) { root->right = rightRotate(root->right);
  return leftRotate(root);
 }
 return root;
}
// Print the tree
void printPreOrder(struct Node *root) {
```

www.binils.com

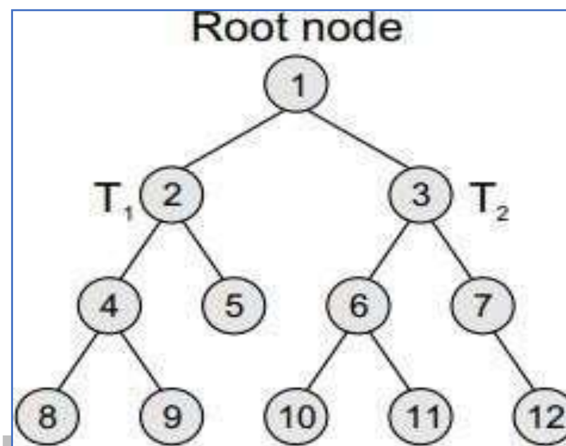
```
if (root != NULL) {

  printf("%d ", root->key);

  printPreOrder(root->left);

  printPreOrder(root-

  >right);

} }
int main() {

 struct Node *root =
 NULL;

 root = insertNode(root,
 2);

 root = insertNode(root,
 1); root =
 insertNode(root, 7);

 root = insertNode(root,
 4);

 root = insertNode(root,
 5);

 root = insertNode(root,
 3);

 root = insertNode(root,
 8); printPreOrder(root);
 root = deleteNode(root,
 3);
 printf("\nAfter deletion:
 ");
 printPreOrder(root);
 return 0;
 }
```

www.binils.com

BINARY TREES

A binary tree is a data structure that is defined as a collection of elements called nodes. In a binary tree, the topmost element is called the root node, and each node has 0, 1, or at the most 2 children.

- Every node contains a data element, a left pointer which points to the left child, and a right pointer which points to the right child. The root element is pointed by a 'root' pointer. If root = NULL, then it means the tree is empty.
- In the figure, R is the root node and the two trees T1 and T2 are called the left and right sub-trees of R. T1 is said to be the left successor of R. Likewise, T2 is called the right successor of R.

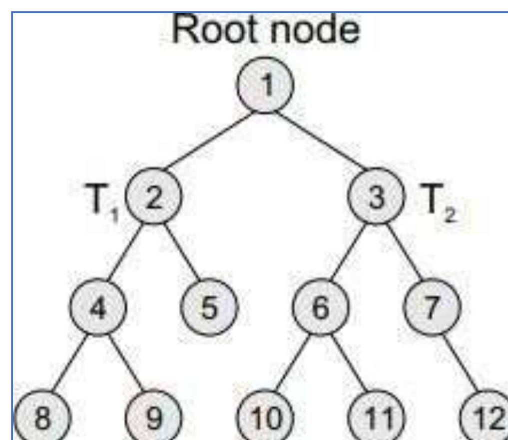


www.binils.com

TERMINOLOGY

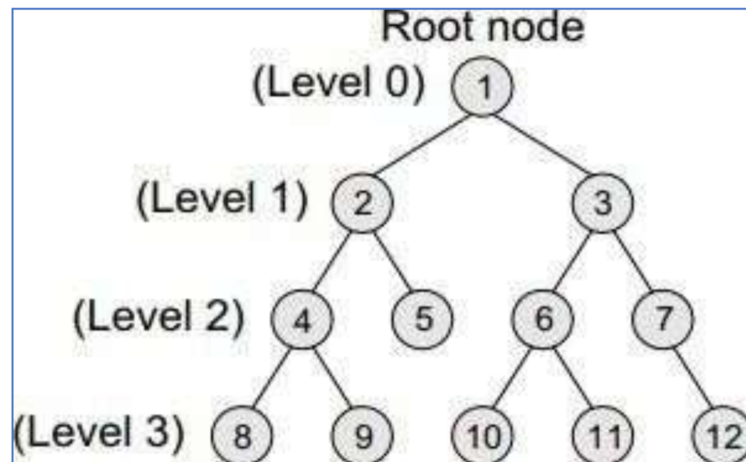
Parent

If N is any node in T that has left successor S1 and right successor S2, then N is called the parent of S 1 and S2. Correspondingly, S1 and S2 are called the left child and the right child of N. Every node other than the root node has a parent



Level number

Every node in the binary tree is assigned a level number. The root node is defined to be at level 0. The left and the right child of the root node have a level number 1. Similarly, 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

www.binils.com

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 that node.

- Out-degree

  Out-degree of a node is the number of edges leaving that node.

It is equal to the number of children that a node has. The degree of a leaf node is zero. For example, in above tree, degree of node 4 is 2, degree of node 5 is zero and degree of node 7 is 1.

Path

A sequence of consecutive edges. For example, in above Fig., the path from the root node to the node 8 is given as: 1, 2, 4, and 8.

Sibling

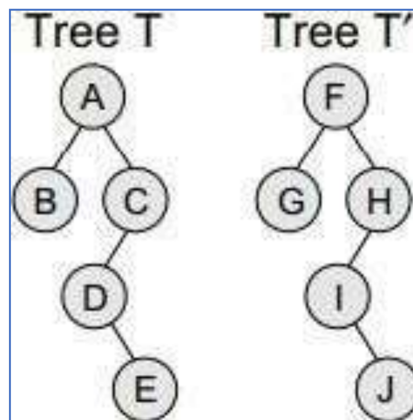
All nodes that are at the same level and share the same parent are called siblings (brothers). For example, nodes 2 and 3; nodes 4 and 5; nodes 6 and 7; nodes 8 and 9; and nodes 10 and 11 are siblings.

Leaf node

A node that has no children is called a leaf node or a terminal node. The leaf nodes in the tree are: 8, 9, 5, 10, 11, and 12.

Similar binary trees

Two binary trees T and T' are said to be similar if both these trees have the same structure. Figure shows two similar binary trees.



Edge

It is the line connecting a node N to any of its successors. A binary tree of n nodes has exactly n - 1 edges because every node except the root node is connected to its parent via an edge.

www.binils.com

Depth

The depth of a node N is given as the length of the path from the root R to the node N. The depth of the root node is zero.

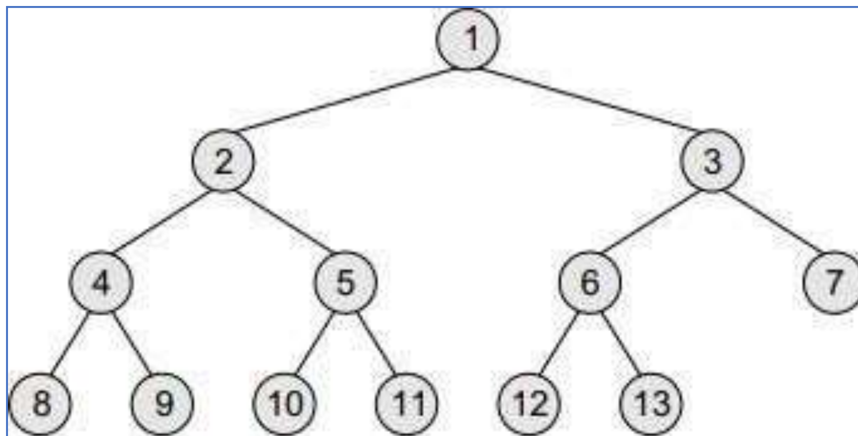
Height of a tree

It is the total number of nodes on the path from the root node to the deepest node in the tree. A tree with only a root node has a height of 1.

COMPLETE BINARY TREES

- A complete binary tree is a binary tree that satisfies two properties. First, in a complete binary tree, every level, except possibly the last, is completely filled. Second, all nodes appear as far left as possible.
- In a complete binary tree Tn , there are exactly n nodes and level r of T can have at most 2r nodes.

Figure shows a complete binary tree.



Above tree has exactly 13 nodes.

- The formula can be given as—if K is a parent node, then its left child can be calculated as 2× K and its right child can be calculated as 2 × K + 1.
- For example, the children of the node 4 are 8 (2 × 4) and 9 (2 × 4 + 1). Similarly, the parent of the node K can be calculated as | K/2 |. Given the node 4, its parent can be calculated as | 4/2 | = 2.
- The height of a tree Tn having exactly n nodes is

given as: Hn = | log2 (n + 1) |

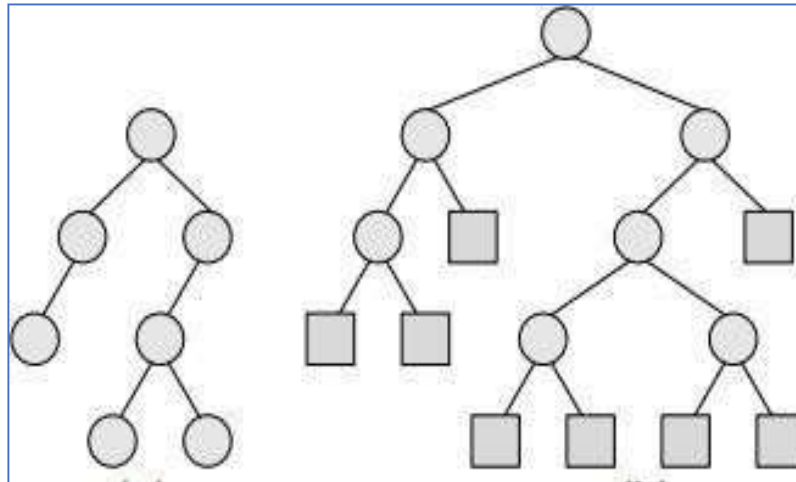
www.binils.com

NOTE: In Fig. 9.7, level 0 has 20 = 1 node, level 1 has 21 = 2 nodes, level 2 has 22 = 4 nodes, level 3 has 6 nodes which is less than the maximum of 23 = 8 nodes.

Extended Binary Trees

- 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.
- In an extended binary tree, nodes having two children are called internal nodes and nodes having no children are called external nodes.
- In Given Fig., the internal nodes are represented using circles and the external nodes are represented using squares.

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



Representation of Binary Trees in the Memory

In the computer's memory, a binary tree can be maintained either by using a linked representation or by using a sequential representation.

www.binils.com

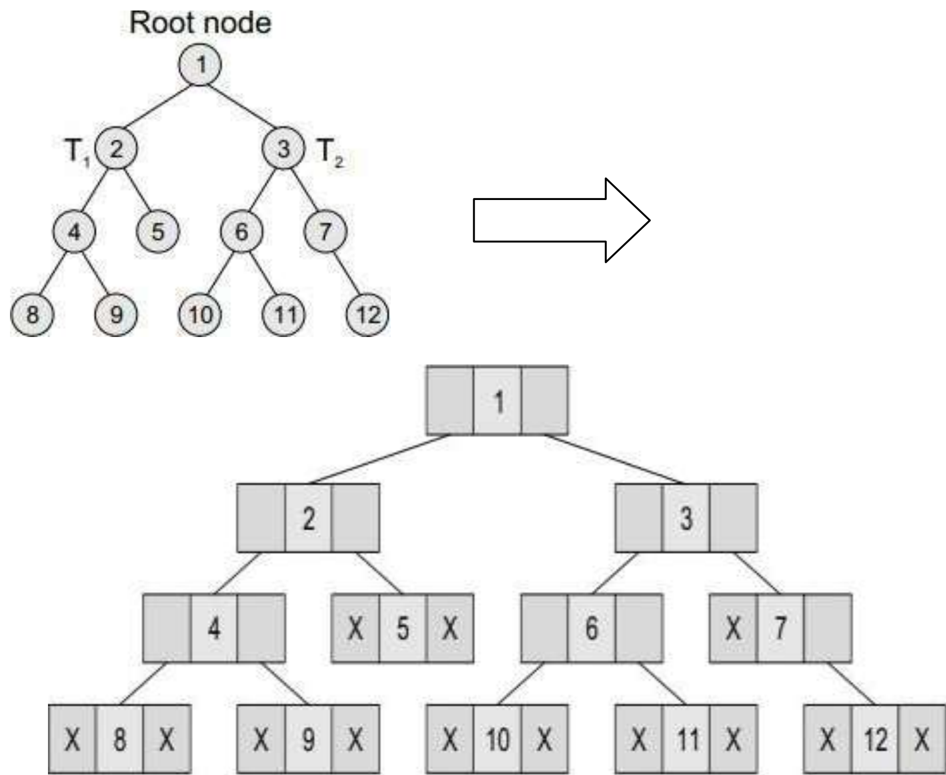
1. Linked representation of binary trees

In the linked representation of a binary tree, every node will have three parts: the data element, a pointer to the left node, and a pointer to the right node.

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

- Every binary tree has a pointer ROOT, which points to the root element (topmost element) of the tree.
- If ROOT = NULL, then the tree is empty.

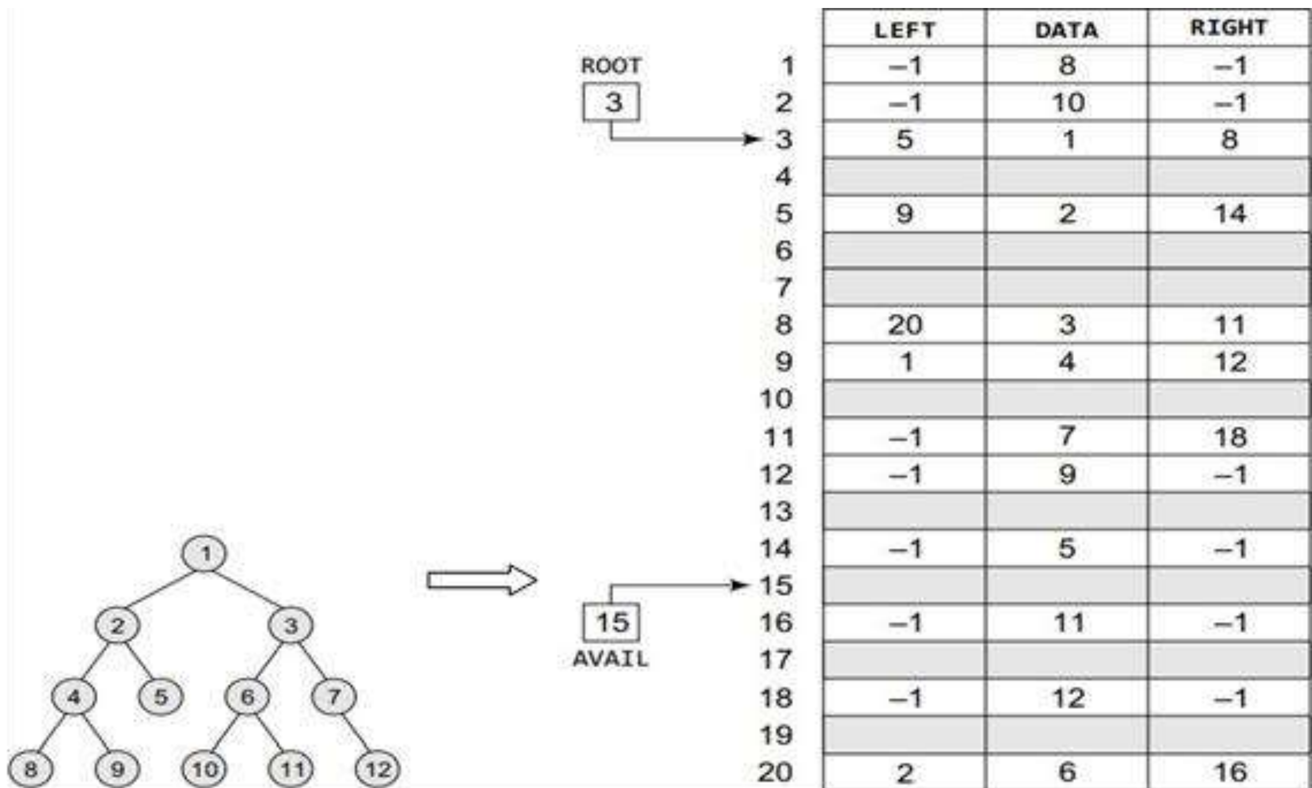
Consider the binary tree given. The schematic diagram of the linked representation of the binary tree is shown in Fig.



www.binils.com

In Fig, the left position is used to point to the left child of the node or to store the address of the left child of the node. The middle position is used to store the data. Finally, the right position is used to point to the right child of the node or to store the address of the right child of the node. Empty sub-trees are represented using X (meaning NULL).

Look at the tree given. Note how this tree is represented in the main memory using a linked list



	LEFT	DATA	RIGHT
1	–1	8	–1
2	–1	10	–1
3	5	1	8
4			
5	9	2	14
6			
7			
8	20	3	11
9	1	4	12
10			
11	–1	7	18
12	–1	9	–1
13			
14	–1	5	–1
15			
16	–1	11	–1
17			
18	–1	12	–1
19			
20	2	6	16

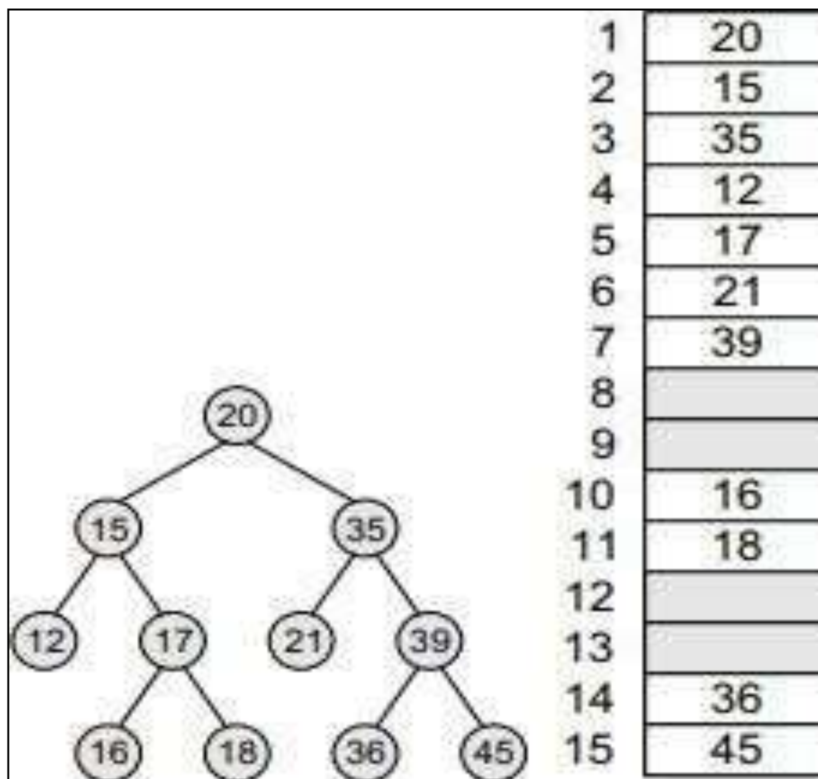
www.binils.com

2. Sequential representation of binary trees

Sequential representation of trees is done using single or one-dimensional arrays. Though it is the simplest technique for memory representation, it is inefficient as it requires a lot of memory space.

A sequential binary tree follows the following rules:

- A one-dimensional array, called TREE, is used to store the elements of tree.
- The root of the tree will be stored in the first location. That is, TREE [1] will store the data of the root element.
- The children of a node stored in location K will be stored in locations (2 × K) and (2× K+1).
- The maximum size of the array TREE is given as (2h-1), where h is the height of the tree.
- An empty tree or sub-tree is specified using NULL. If TREE [1] = NULL, then the tree is empty.

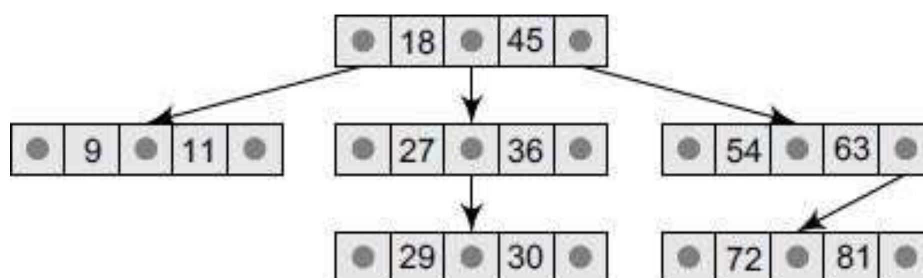


This shows a binary tree and its corresponding sequential representation. The tree has 11 nodes and its height is 4

www.binils.com

MULTI-WAY SEARCH TREES

We have discussed that every node in a binary search tree contains one value and two pointers, left and right, which point to the node's left and right sub-trees, respectively. The same concept is used in an M-way search tree which has M – 1 values per node and M subtrees. In such a tree, M is called the degree of the tree. Note that in a binary search tree M = 2, so it has one value and two sub-trees. In other words, every internal node of an M-way search tree consists of pointers to M sub-trees and contains M – 1 keys, where M > 2.



M-way search tree of order 3

www.binils.com

B TREES

- Btreeisaspecialized M-waytreethatiswidelyused fordiskaccess. Btreeoforder m can have a maximum of m– 1 keys and m pointers to its sub-trees. B tree may contain a large number of key values and pointers to sub- trees. Storing a large number ofkeys in a single node keeps the heightofthe treerelatively small.
- B tree is designed to store sorted data and allows search, insertion, and 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 the 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.

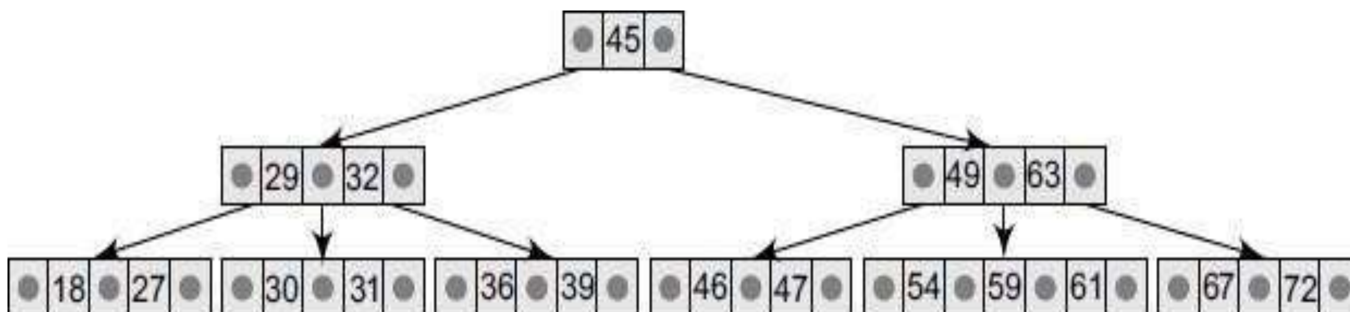
2. Every node in the B tree except the root node and leaf nodes has at least (minimum) m/2 children. Thiscondition helps to keep thetreebushy so thatthe pathfromtheroot node to the leaf is very short, even in a tree that stores a lot ofdata.

3. The root node has at least two children if it is not aterminal (leaf) node.

4. All leaf nodes are at the same level.

1. Searching for an Element in a B Tree

Searching for an element in a B tree is similar to that in binary search trees. Consider the B tree iven in Fig. To search for 59



Step 1. we begin at the root node. The root node has a value 45 which is less than 59.

Step 2. So, we traverse in the right sub-tree. The right sub-tree of the root node has two key values, 49 and 63. Since 49 < 59 < 63,

Step 3. We traverse the right sub-tree of 49, that is, the left sub-tree of 63. This sub-tree has three values, 54, 59, and 61. On finding the value 59, the search is successful.

www.binils.com

The running time of the search operation depends upon the height of the tree, the algorithm to search for an element in a B tree takes O(logt n) time to execute.

2. Inserting a New Element in a B Tree

In a B tree, all insertions are done at the leaf node level. A new value is inserted in the B tree using the algorithm given below.

Procedure:

Step 1. Search the B tree to find the leaf node where the new key value should be inserted.

Step 2. If the leaf node is not full, that is, it contains less than m–1 key values, then insert the new element in the node keeping the node's elements ordered.

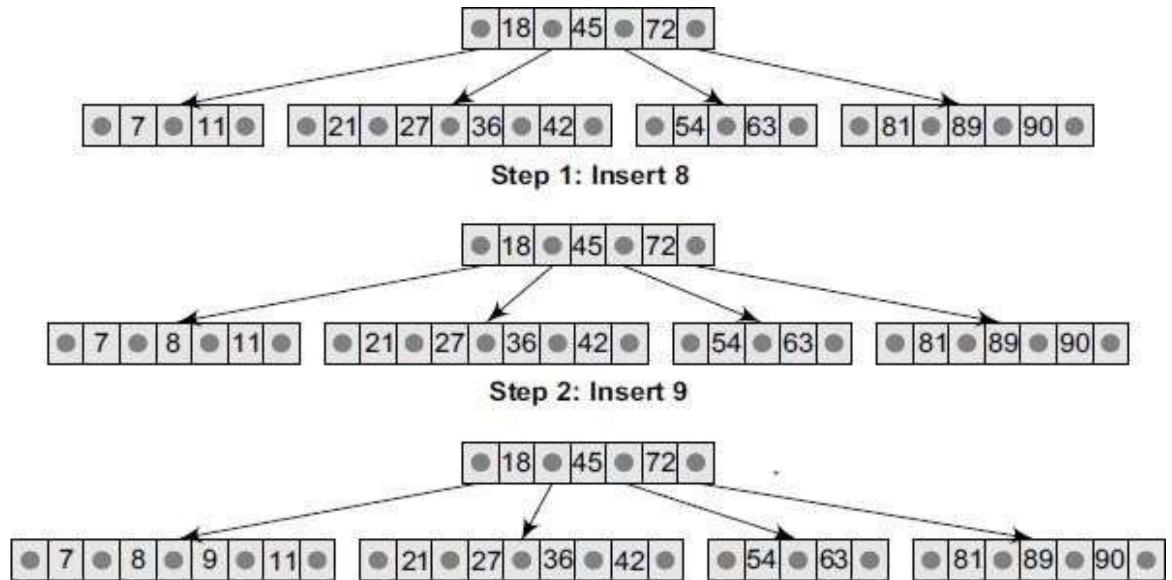
Step 3. If the leaf node is full, that is, the leaf node already contains m–1 key values, then

(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 the parent's node is already full, then split the parent node by following the same steps

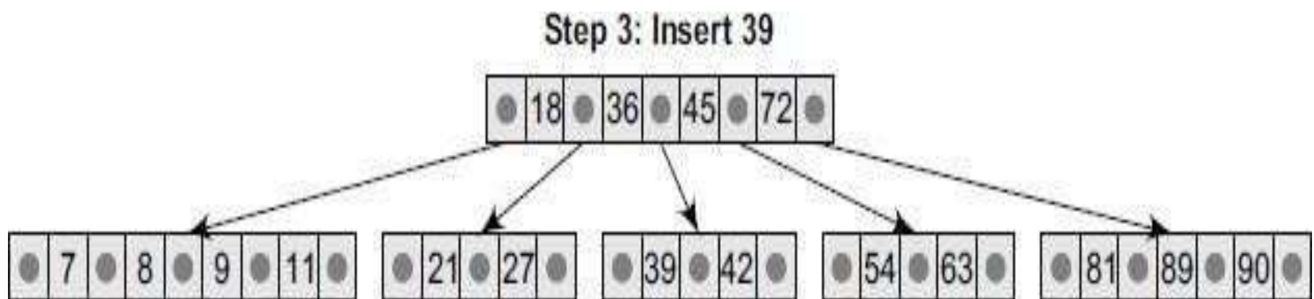
Example 1 Lookatthe Btreeoforder 5 givenbelow and insert 8, 9, 39, and 4 into it.



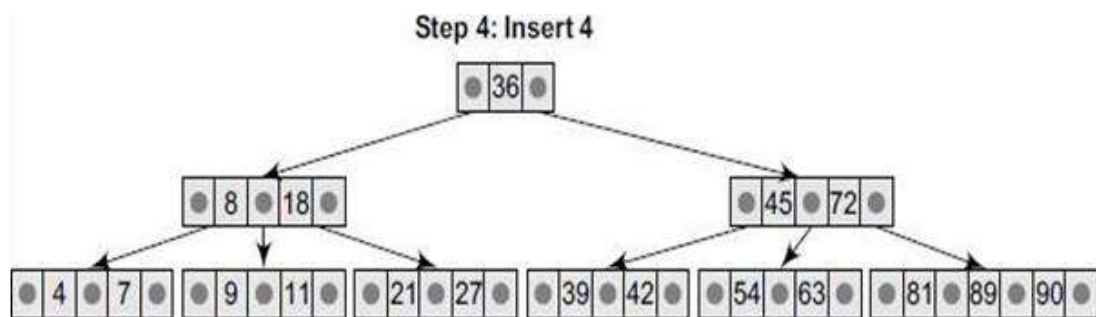
www.binils.com

Till now, we have easily inserted 8and 9 in the tree because the leaf nodes were not full. But now, the node in which 39 should be inserted is already full as it contains four values. Here we split the nodes to form two separate nodes. But before splitting, arrange the key values in order (including the new value).

The ordered set of values isgivenas21, 27, 36,39, and42. The median value is 36, so push 36 into its parent's node and split the leaf nodes



Now the node in which 4 should be inserted is already full as it contains four key values. Here we split the nodes to form two separate nodes. But before splitting, we arrange the key values in order (including the new value). The ordered set of values is given as 4, 7, 8, 9, and 11. The median value is 8, so we push 8 into its parent's node and split the leaf nodes. But again, we see that the parent's node is already full, so we split the parent node using the same procedure.



3. Deleting an Element from a B Tree

www.binils.com

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.

Step 1. Locate the leaf node which has to be deleted.

Step 2. If the leaf node contains more than the minimum number of key values (more than m/2 elements), then delete the value.

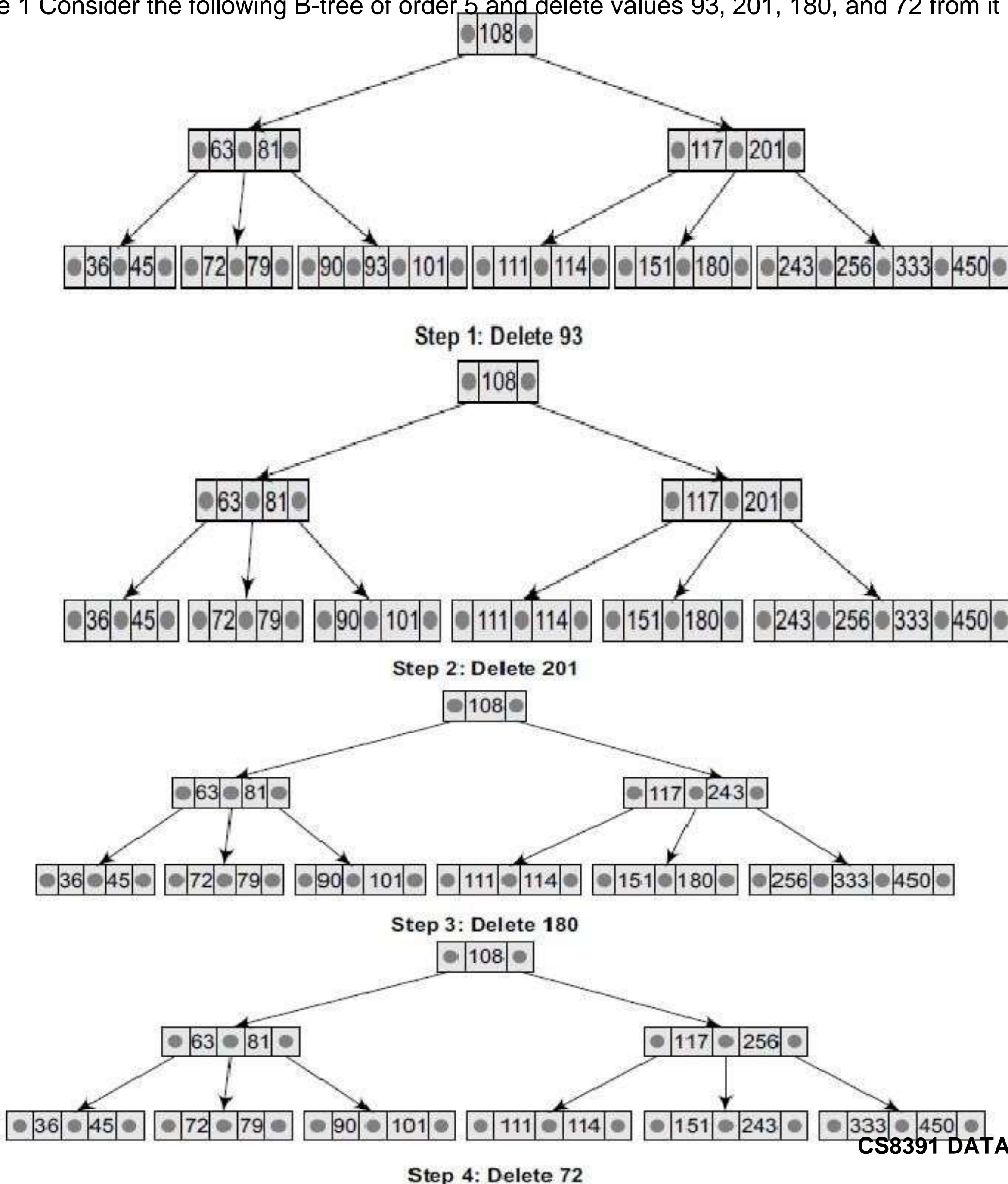
Step 3. Else if the leaf node does not contain m/2 elements, then fill the node by taking an element either from the left or from the right sibling.

(a) If the left sibling has more than the minimum number of key values, push its largest key into its 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 sibling 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.

Step 4. Else, if both left and right siblings contain only the minimum number of elements, then create a new leaf node by combining the two leaf nodes and the intervening element of the parent

Example 1 Consider the following B-tree of order 5 and delete values 93, 201, 180, and 72 from it



Step 1: Delete 93

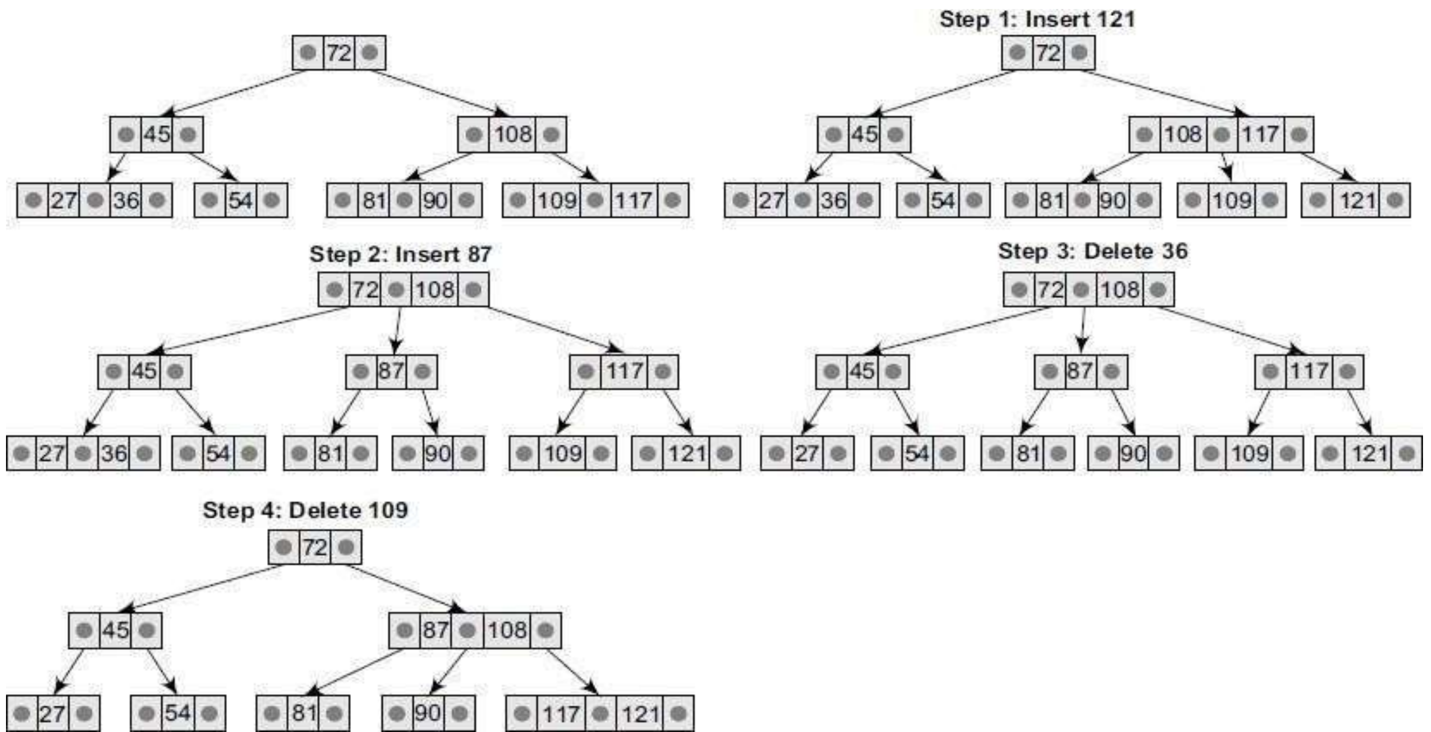
Step 2: Delete 201

Step 3: Delete 180

Step 4: Delete 72

Example 2 Consider the B tree of order 3 given below and perform the following operations:

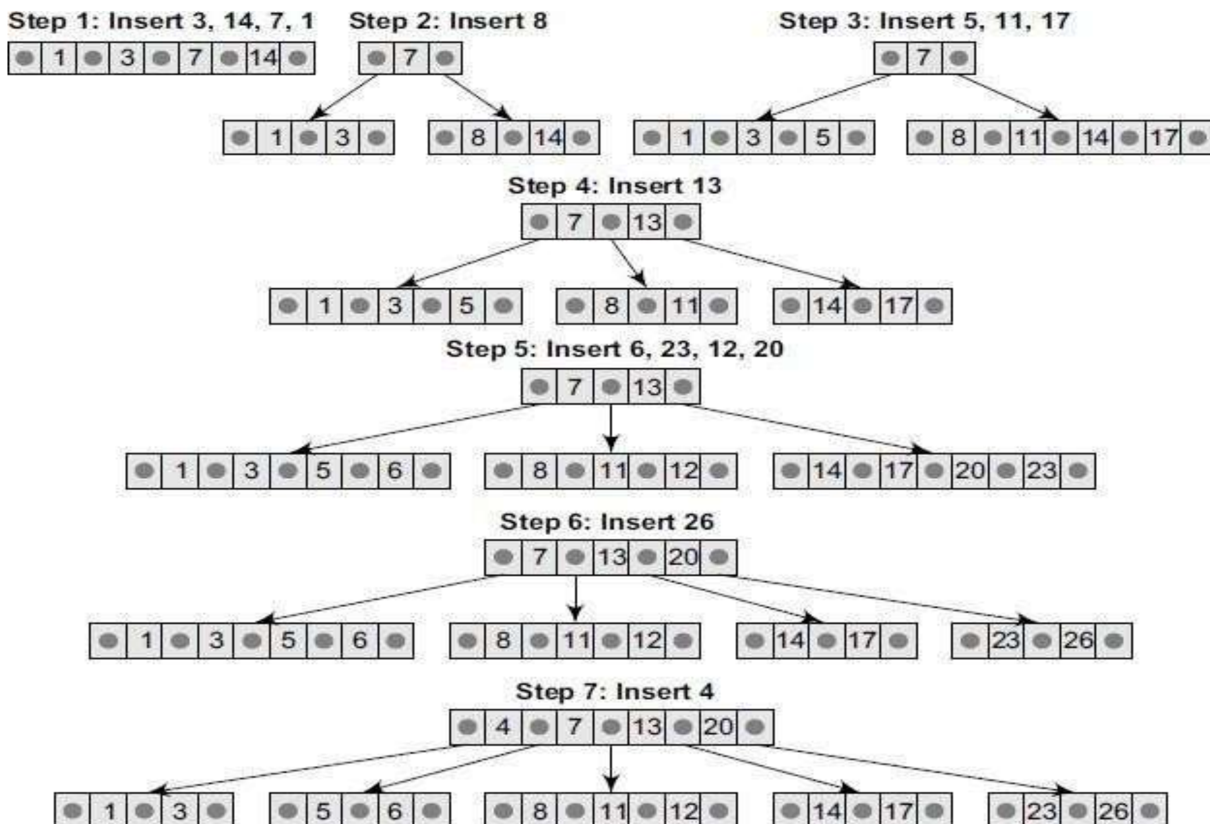
(a) insert 121, 87 and then (b) delete 36, 109.

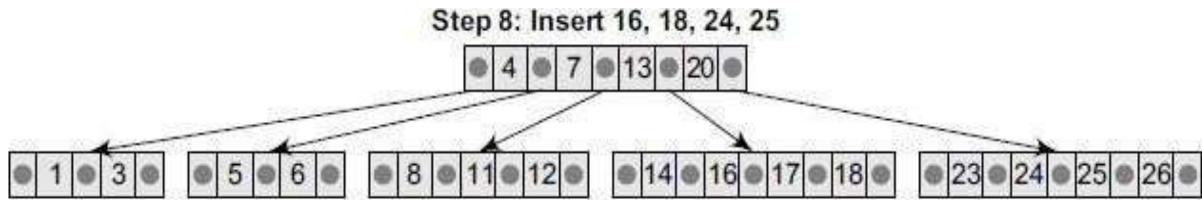


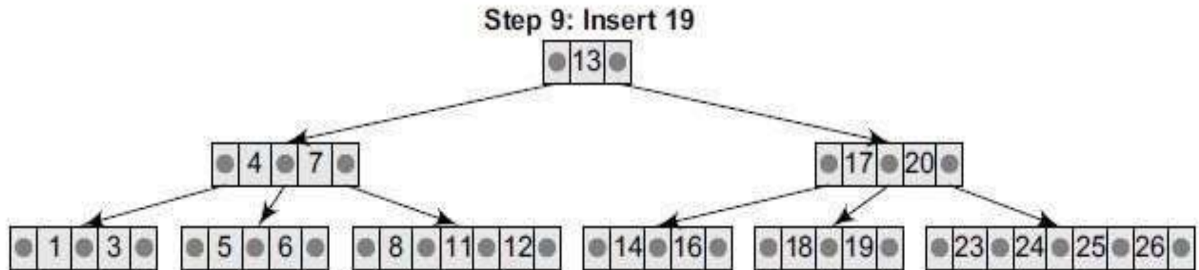
www.binils.com

Example 11.4 Create a B tree of order 5 by inserting the following elements: 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20,

26, 4, 16, 18, 24, 25, and 19

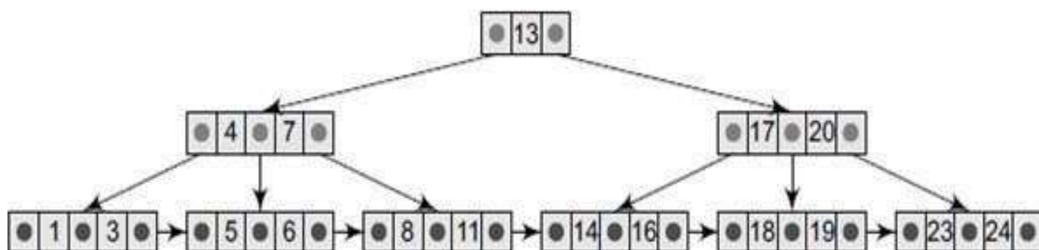






B+ TREES

- A B+ tree is a variant of a B tree which stores sorted data in a way that allows for efficient insertion, retrieval, and removal of records, each of which is identified by a key. While a B tree can store both keys and records in its interior nodes, a B+ tree, in contrast, stores all the records at the leaf level of the tree; only keys are stored in the interior nodes.
- The leaf nodes of a B+ tree are often linked to one another in a linked list. This has an added advantage of making the queries simpler and more efficient.
- Typically, B+ trees are used to store large amounts of data that cannot be stored in the main memory. With B+ trees, the secondary storage (magnetic disk) is used to store the leaf nodes of trees and the 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 allows us to traverse the tree from the root down to the leaf node that stores the desired data item. Figure shows a B+ tree of order 3.



Many database systems 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 of as a multi-level index in which the leaves make up a dense index and the non-leaf nodes make up a sparse index.

The advantages of B+ trees can be given as follows:

1. Records can be fetched in equal number of disk accesses
2. It can be used to perform a wide range of queries easily as leaves are linked to nodes at the upper level
3. Height of the tree is less and balanced
4. Supports both random and sequential access to records
5. Keys are used for indexing

1. Inserting 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 the 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, in turn, to split.
- In fact, all the nodes on the path from a leaf to the root may split 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.

The steps to insert a new node in a B+ Tree are summarized

below Step 1: Insert the new node as the leaf node.

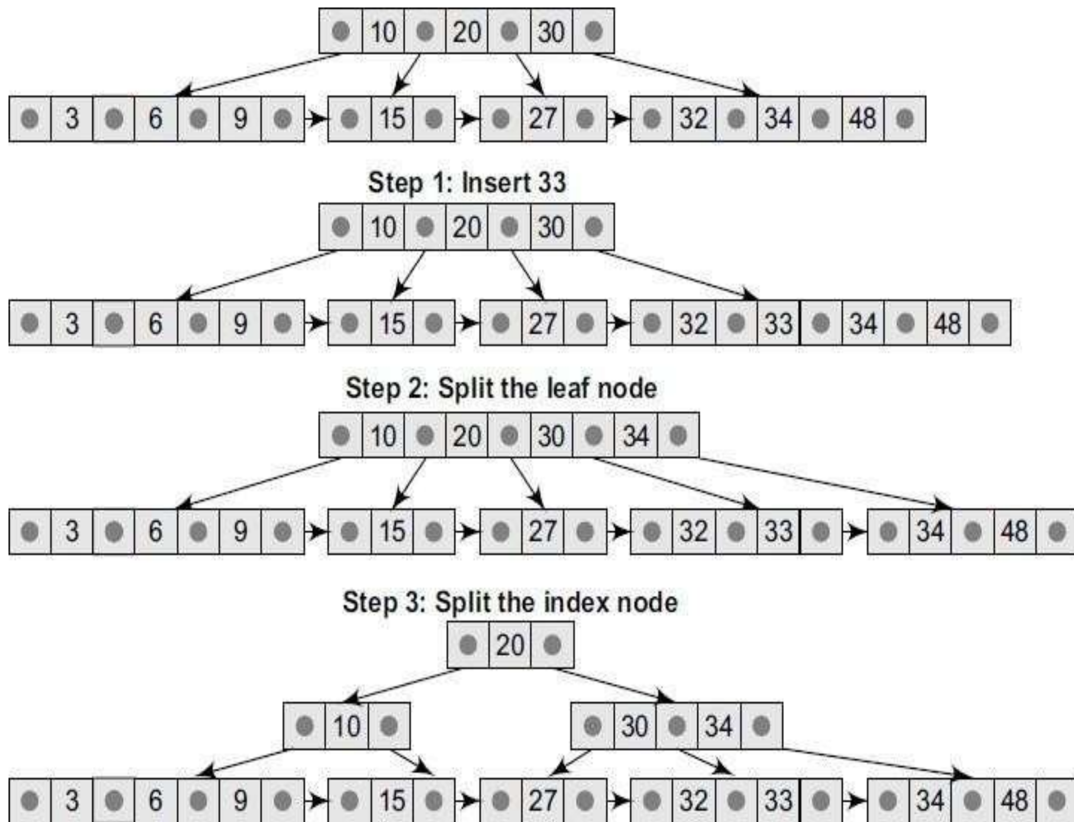
Step 2: If the leaf node overflows, split the node and copy the middle element to next index

node. Step 3: If the index node overflows, split that node and move the middle element to

next index page.

www.binils.com

Example 1. Consider the B+ tree of order 4 given and insert 33 in it.



2 Deleting an Element from a B+ Tree

www.binils.com

- As in B trees, deletion is always done from a leaf node. If deleting a data element leaves that node empty, then the neighboring nodes are examined and merged with the under full 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 the root. This leads to shrinking of the tree by one level.

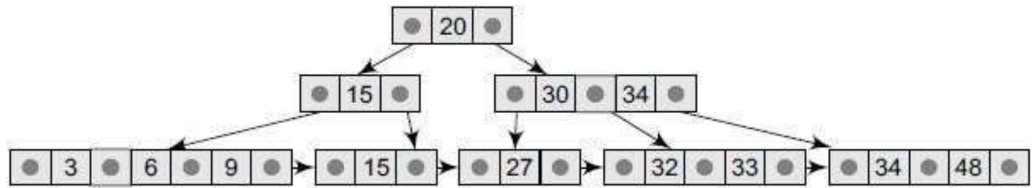
The steps to delete a node from a B+ tree are summarized

below Step 1: Delete the key and data from the leaves.

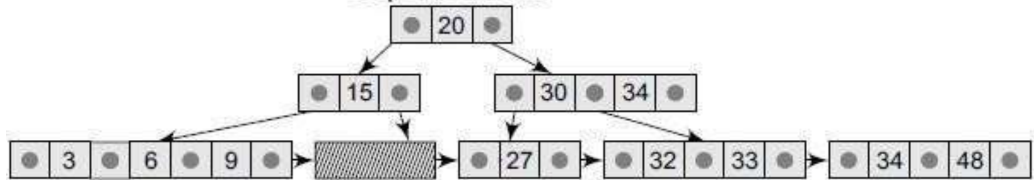
Step 2: If the leaf node underflows, merge that node with the sibling and delete the key in between them.

Step 3: If the index node underflows, merge that node with the sibling and move down the key in between them.

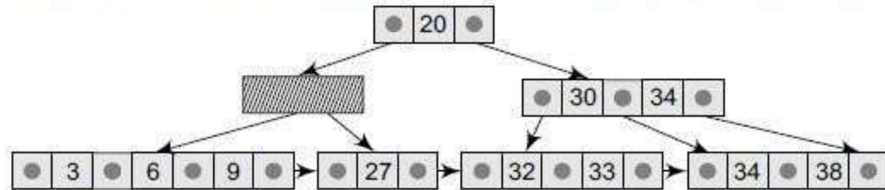
Example 1: Consider the B+ tree of order 4 given below and delete node 15 from it.



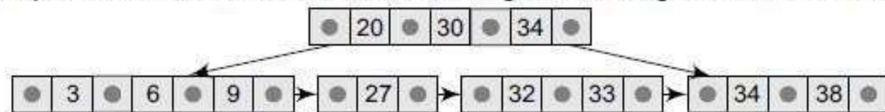
Step 1: Delete 15



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



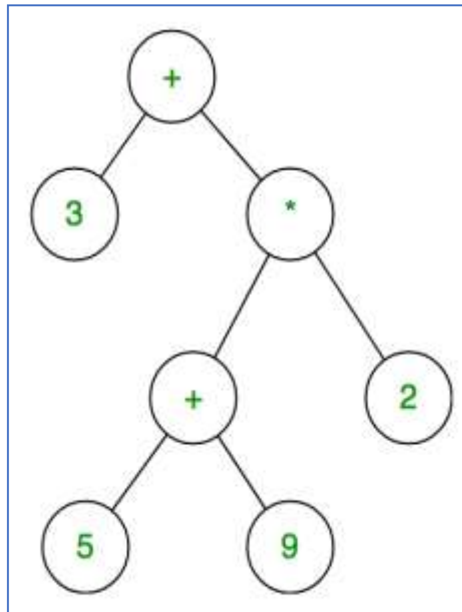
Step 3: Now index node underflows, so merge with sibling and delete the node



www.binils.com

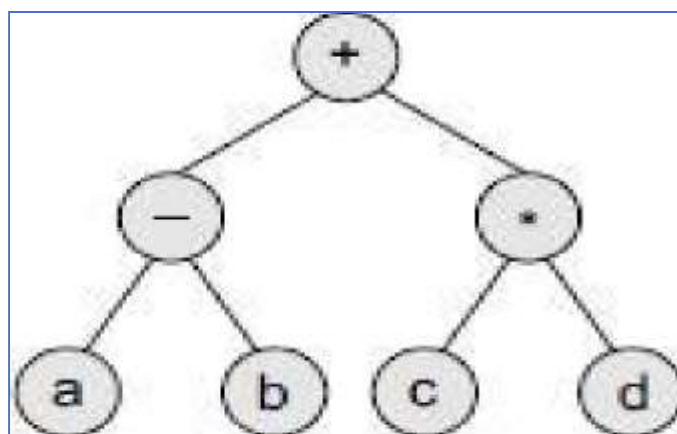
Expression Trees

The expression tree is a binary tree in which each internal node corresponds to the operator and each leaf node corresponds to the operand so for example expression tree for 3 + ((5+9)*2) would be:



www.binils.com

Binary trees are widely used to store algebraic expressions. For example, consider the algebraic expression given as: Exp = (a – b) + (c * d)



Evaluating the expression represented by an expression tree:

Let t be the expression

tree If t is not null then

If t.value is operand

then Return

t.value

A = solve(t.left)

B =

solve(t.right)

// calculate applies operator 't.value'

// on A and B, and returns

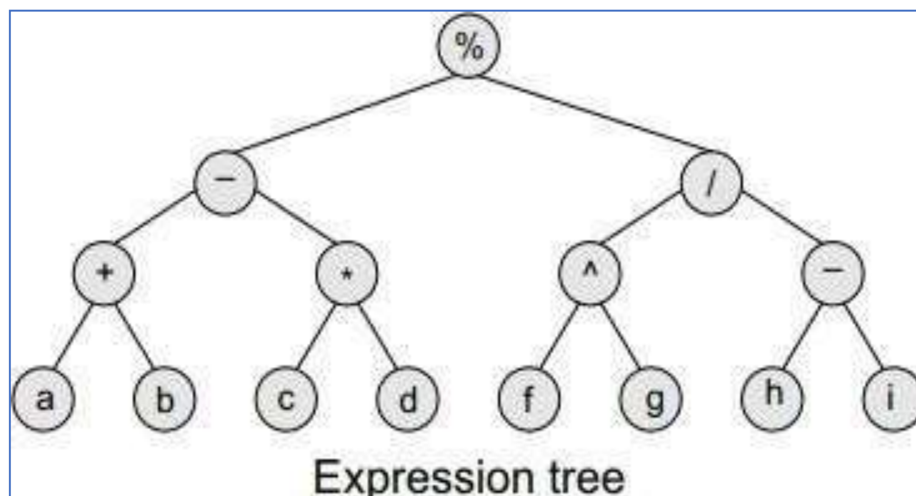
value Return calculate(A, B,

t.value)

www.binils.com

Example 1. Given an expression, Exp = ((a + b) – (c * d)) % ((e ^f) / (g – h)), construct the corresponding

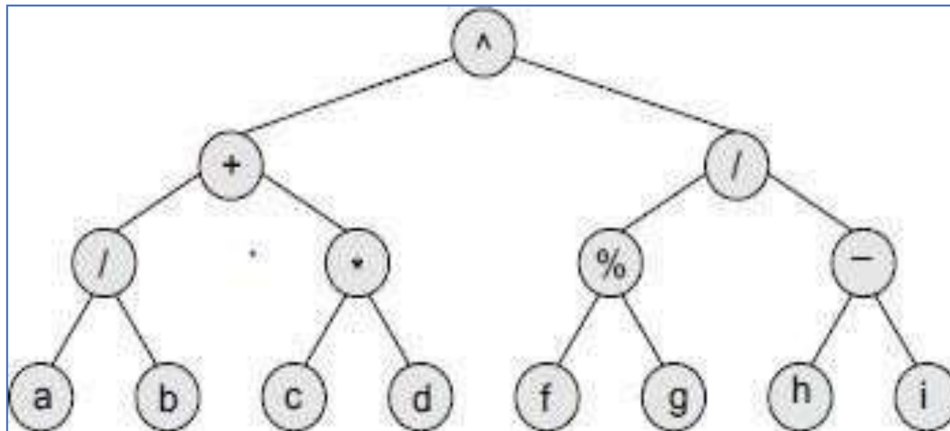
binary tree. Solution



Expression tree

Example 2. Given the binary tree, write down the expression that it

represents. Solution



Construction of Expression Tree:

Now for constructing an expression tree we use a stack. We loop through input expression and do the following for every character.

www.binils.com

- If a character is an operand push that into the stack
- If a character is an operator pop two values from the stack make them its child and push the current node again.

In the end, the only element of the stack will be the root of an expression tree.

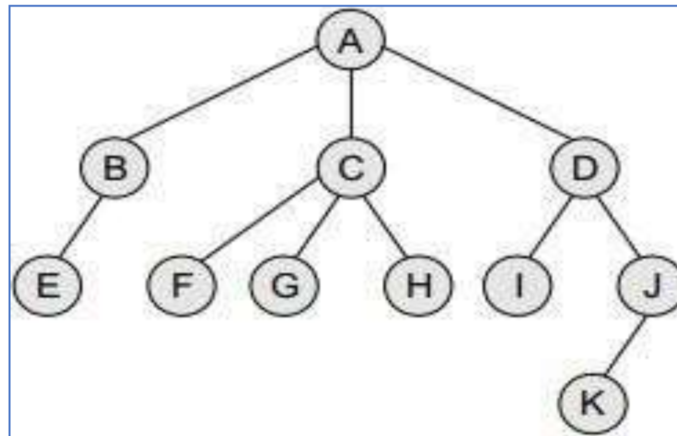
Creating a Binary Tree from a General Tree

The rules for converting a general tree to a binary tree are given below. Note that a general tree is converted into a binary tree and not a binary search tree.

Rule 1: Root of the binary tree = Root of the general tree

Rule 2: Left child of a node = Leftmost child of the node in the binary tree in the general tree

Rule 3: Right child of a node in the binary tree = Right sibling of the node in the general tree



Convert the given general tree into a binary tree

Now let us build the binary tree.

Step 1: Node A is the root of the general tree, so it will also be the root of the binary tree.

Step 2: Left child of node A is the leftmost child of node A in the general tree and right child of node A is the right sibling of the node A in the general tree. Since node A has no right sibling in the general tree, it has no right child in the binary tree.

Step 3: Now process node B. Left child of B is E and its right child is C (right sibling in general tree).

Step 4: Now process node C. Left child of C is F (leftmost child) and its right child is D (right sibling in generaltree).

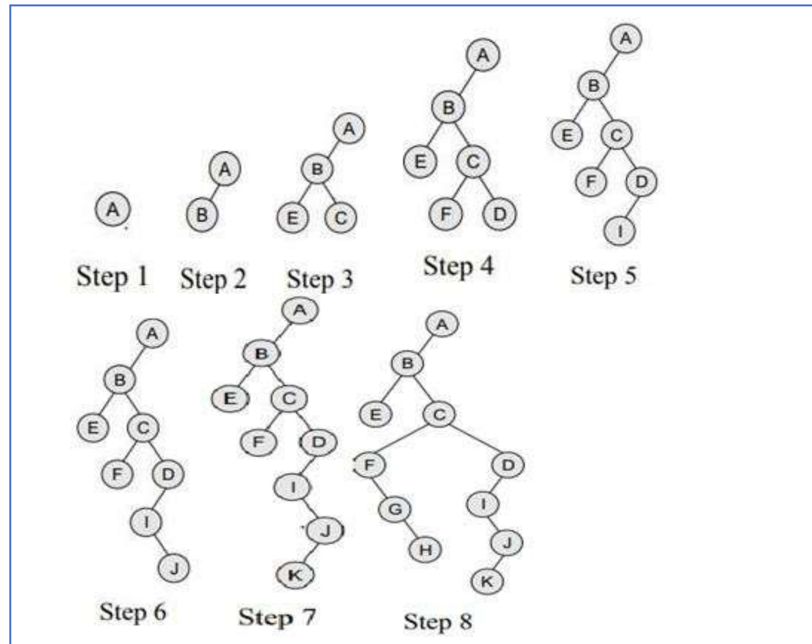
Step 5: Now process node D. Left child of D is I (leftmost child). There will be no right child of D because it has no right sibling in the general tree.

Step 6: Now process node I. There will be no left child of I in the binary tree because I has no left child in the general tree. However, I has a right sibling J, so it will be added as the right child of I.

Step 7: Now process node J. Left child of J is K (leftmost child). There will be no right child of J because it has no right sibling in the general tree.

www.binils.com

Step 8: Now process all the unprocessed nodes (E, F, G, H, K) in the same fashion, so the resultant binary tree can be given as follows.



www.binils.com

HEAPS

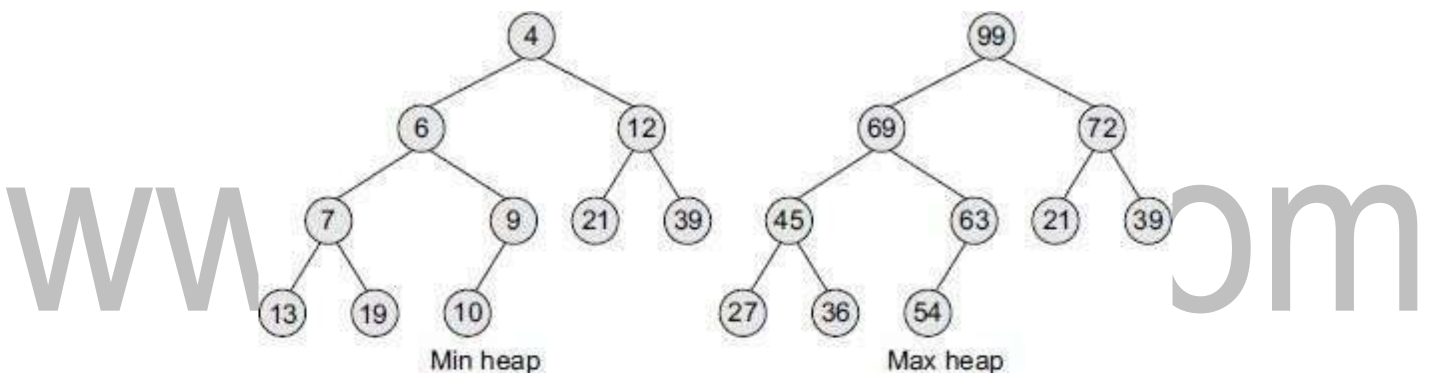
BINARY HEAPS

A binary heap is a complete binary tree in which every node satisfies the heap property which

states that: If B is a child of A, then key(A) ≥ key(B)

- This implies that elements at every node will be either greater than or equal to the element at its left and right child. Thus, the root node has the highest key value in the heap. Such a heap is commonly known as a max- heap.
- Alternatively, elements at every node will be either less than or equal to the element at its left and right child.

  Thus, the root has the lowest key value. Such a heap is called a min-heap.



The properties of binary heaps are given as follows:

Since a heap is defined as a complete binary tree, all its elements can be stored sequentially in an array. It follows the same rules as that of a complete binary tree. That is, if an element is at position i in the array, then its left child is stored at position 2i and its right child at position 2i+1. Conversely, an element at position ihas its parent stored at position i/2.

- Being a complete binary tree, all the levels of the tree except the last level are completely filled.
- The height of a binary tree is given as log2n, where n is the number of elements.
- Heaps (also known as partially ordered trees) are a very popular data structure for implementing priority queues.

OPERATIONS:

1.Insertio

n

2.Deletio

n

Inserting a New Element in a Binary Heap

Consider a max heap H with n elements. Inserting a new value into the heap is done in the following two steps:

1. Add the new value at the bottom of H in such a way that H is still a complete binary tree but not necessarily a heap.

2. Let the new value rise to its appropriate place in H so that H now becomes a heap as well. To do this, compare the new value with its parent to check if they are in the correct order. If they are, then the procedure halts, else the new value and its parent's value are

www.binils.com

swapped and Step 2 is repeated.

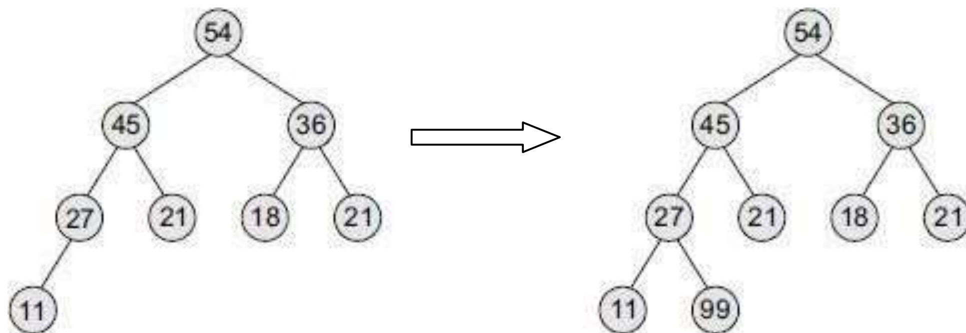
```
Step 1: [Add the new value and set its POS]
        SET N = N + 1, POS = N
Step 2: SET HEAP[N] = VAL
Step 3: [Find appropriate location of VAL]
        Repeat Steps 4 and 5 while POS > 1
Step 4:     SET PAR = POS/2
Step 5:     IF HEAP[POS] <= HEAP[PAR],
            then Goto Step 6.
            ELSE
                  SWAP HEAP[POS], HEAP[PAR]
                  POS = PAR
            [END OF IF]
      [END OF LOOP]
Step 6: RETURN
```

Algorithm to insert an element in a max heap

Example 1 Consider the max heap given in Fig. and insert

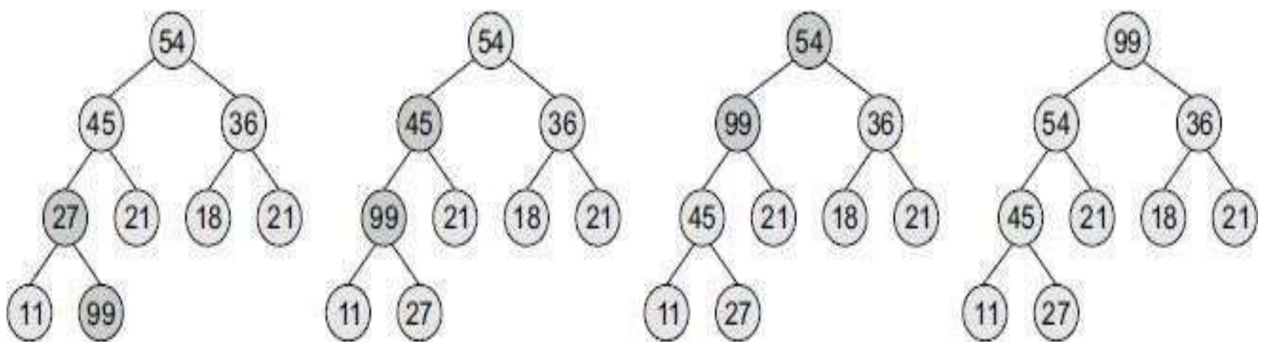
99 in it. Solution

The first step says that insert the element in the heap so that the heap is a complete binary tree. So, insert the new value as the right child of node 27 in the heap. This is illustrated in Fig. Now, as per the second step, let the new value rise to its appropriate place in H so that H becomes a heap as well



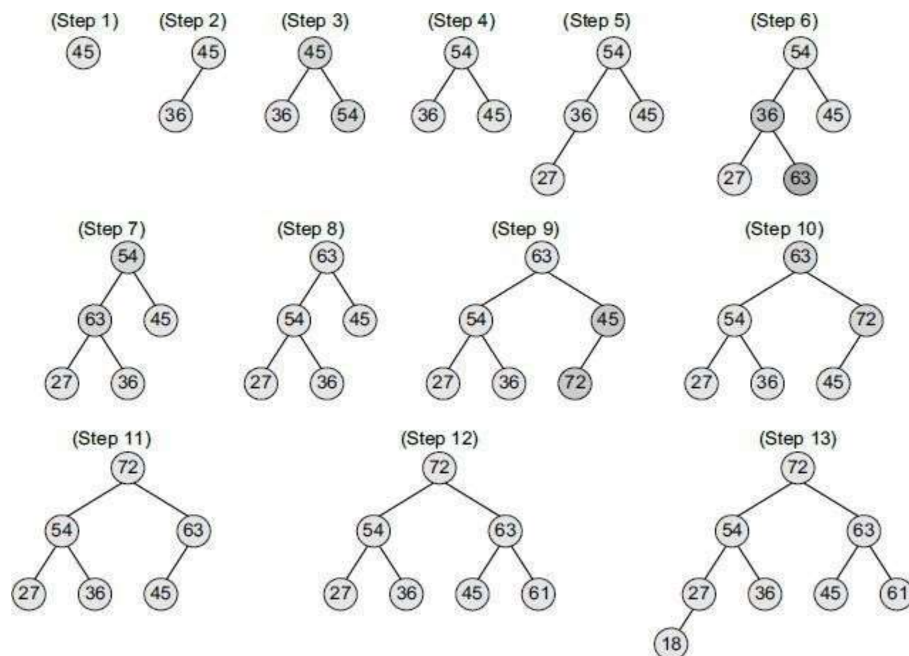
Compare 99 with its parent node value. If it is less than its parent's value, then the new node is in its appropriate place and H is a heap. If the new value is greater than that of its parent's node, then swap the two values. Repeat the whole process until H becomes a heap. This is illustrated in Fig.

www.binils.com



Example 2 Build a max heap H from the given set of numbers: 45, 36, 54, 27, 63, 72, 61, and 18. Also draw the memory representation of the heap.

Solution



www.binils.com

2. Deleting an Element from a Binary Heap

Consider a max heap H having n elements. An element is always deleted from the root of the heap. So, deleting an element from the heap is done in the following three steps:

1. Replace the root node's value with the last node's value so that H is still a complete binary tree but not necessarily a heap.

2. Delete the last node.

3. Sink down the new root node's value so that H satisfies the heap property. In this step, interchange the root node's value with its child node's value (whichever is largest among its children). Here, the value of root node

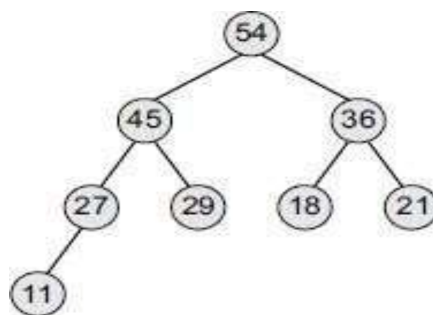
= 54 and the value of the last node = 11. So, replace 54 with 11 and delete the last node.

```
Step 1: [Remove the last node from the heap]
        SET LAST = HEAP[N], SET N = N - 1
Step 2: [Initialization]
        SET PTR = 1, LEFT = 2, RIGHT = 3
Step 3: SET HEAP[PTR] = LAST
Step 4: Repeat Steps 5 to 7 while LEFT <= N
Step 5: IF HEAP[PTR] >= HEAP[LEFT] AND
         HEAP[PTR] >= HEAP[RIGHT]
              Go to Step 8
        [END OF IF]
Step 6: IF HEAP[RIGHT] <= HEAP[LEFT]
              SWAP HEAP[PTR], HEAP[LEFT]
              SET PTR = LEFT
        ELSE
              SWAP HEAP[PTR], HEAP[RIGHT]
              SET PTR = RIGHT
        [END OF IF]
Step 7: SET LEFT = 2 * PTR and RIGHT = LEFT + 1
        [END OF LOOP]
Step 8: RETURN
```

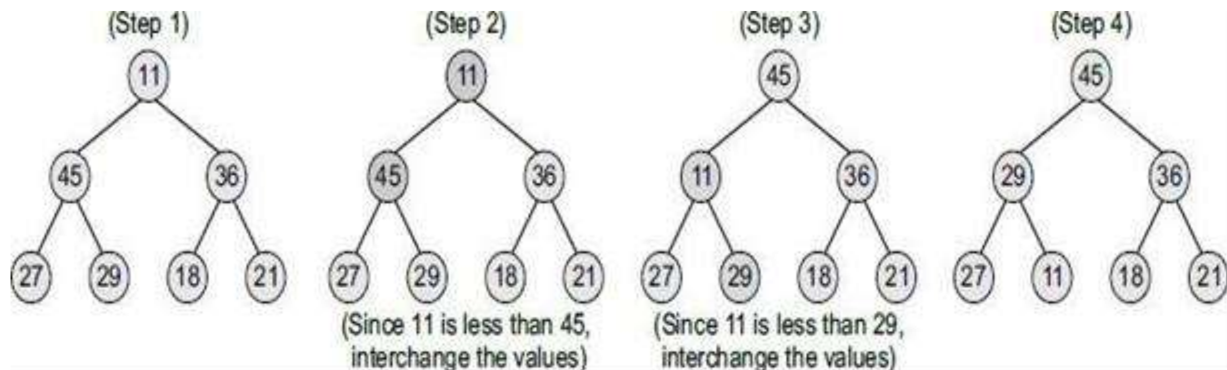
Algorithm to delete the root element from a max heap

www.binils.com

Example 1 Consider the max heap H shown in Fig. 12.8 and delete the root node's value.



Solution



Applications of Binary Heaps

Binary heaps are mainly applied for

1. Sorting an array using heapsort algorithm.
2. Implementing priority queues.

Binary Heap Implementation of Priority Queues

www.binils.com

- A priority queue is similar to a queue in which an item is dequeued (or removed) from the front. However, unlike a regular queue, in a priority queue the logical order of elements is determined by their priority. While the higher priority elements are added at the front of the queue, elements with lower priority are added at the rear.
- Though we can easily implement priority queues using a linear array, but we should first consider the time required to insert an element in the array and then sort it. We need O(n) time to insert an element and at least O(n log n) time to sort the array. Therefore, a better way to implement a priority queue is by using a binary heap which allows both enqueue and dequeue of elements in O(log n) time.

BINOMIAL HEAPS

A binomial heap H is a set of binomial trees that satisfy the binomial heap properties. First, let us discuss what a binomial tree is.

A binomial tree is an ordered tree that can be recursively defined as follows:

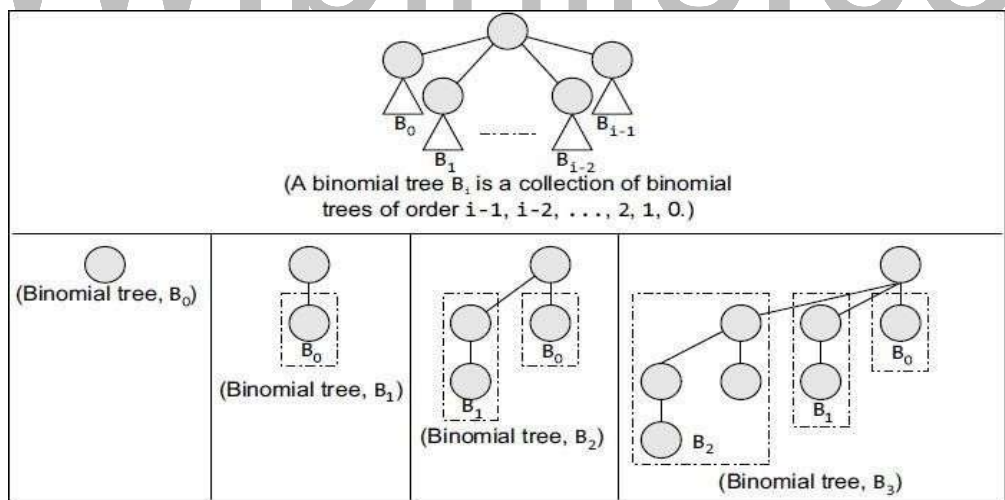
- A binomial tree of order 0 has a single node.

- A binomial tree of order i has a root node whose children are the root nodes of binomial trees of order i–1, i–2, ..., 2, 1, and 0.
- A binomial tree Bi has 2i nodes.

- The height of a binomial tree Bi is i.

Look at Fig. which shows a few binomial trees of different orders. We can construct a binomial tree Bi from two binomial trees of order Bi–1 by linking them together in such a way that the root of one is the leftmost child of the root of another.

www.binils.com



A binomial heap H is a collection of binomial trees that satisfy the following properties:

- Every binomial tree in H satisfies the minimum heap property (i.e., the key of a node is either greater than or equal to the key of its parent).

- There can be one or zero binomial trees for each order including zero order. According to the first property, the root of a heap-ordered tree contains the smallest key in the tree. The second property, on the other hand, implies that a binomial heap H having N nodes contains at most log (N + 1) binomial trees.

3. FIBONACCI HEAPS

A Fibonacci heap is a collection of trees. It is loosely based on binomial heaps. If neither the decrease-value nor the delete operation is performed, each tree in the heap is like a binomial tree. Fibonacci heaps differ from binomial heaps as they have a more relaxed structure, allowing improved asymptotic time bounds.

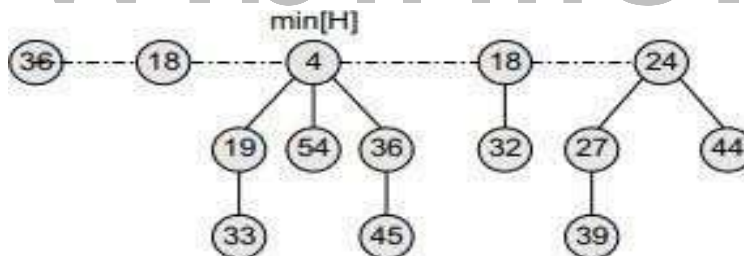
1. Structure of Fibonacci Heaps

Although a Fibonacci heap is a collection of heap-ordered trees, the trees in a Fibonacci heap are not constrained to be binomial trees. That is, while the trees in a binomial heap are ordered, those within Fibonacci heaps are rooted but unordered.

Each node in the Fibonacci heap contains the following pointers:

- a pointer to its parent, and
- a pointer to any one of its children

www.binils.com



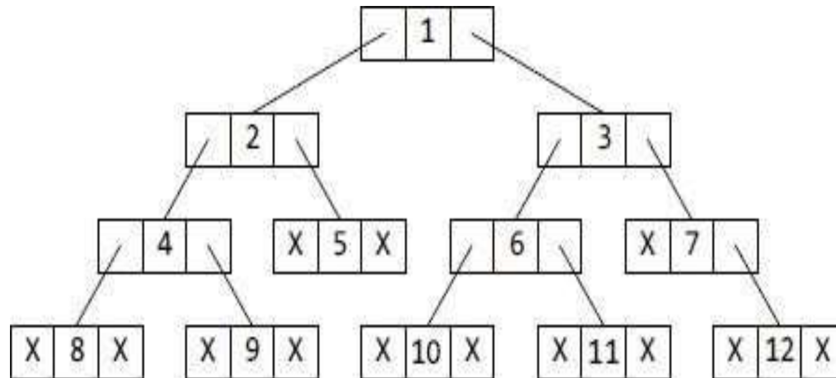
Applications of Heaps

- Heap Implemented priority queues are used in Graph algorithms like Prim's Algorithm and Dijkstra's algorithm.
- Order statistics: The Heap data structure can be used to efficiently find the kth smallest (or largest) element in an array.

THREADED BINARY TREEs

A threaded binary tree is the same as that of a binary tree but with a difference in storing the NULL pointers.

Considerthe linked representationofabinarytreeasgiven in Fig.



www.binils.com

In the linked representation, a number of nodes contain a NULL pointer, either in their left or right fields or in both. This space that is wasted in storing a NULL pointer can be efficiently used to store some other useful piece of information.

For example, the NULL entries can be replaced to store a pointer to the in-order predecessor or the in-order successor of the node. These special pointers are called **threads** and binary trees containing threads are called **threaded trees**. In the linked representation ofa threaded binarytree, threads will be denoted usingarrows.

Types

1. One-way threaded tree
2. Twoway threadedtree

One-way threaded tree

A one-way threaded tree is also called a single-threaded tree. If the thread appears in the left field, then the left field will be made to point to the in-order predecessor of the node. Sucha one-way threaded tree iscalleda left-threaded binary tree.

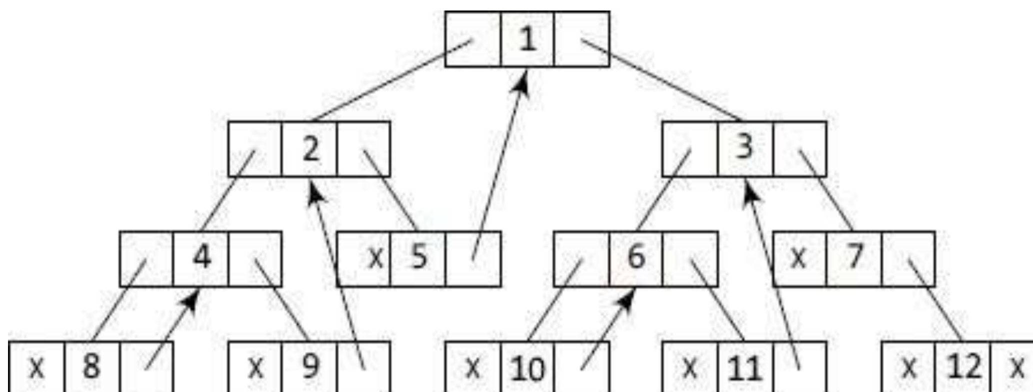
Two-way threaded tree

- On the contrary, if the thread appears in the right field, then it will point to the in- order successor ofthe node. Sucha one-waythreaded tree is called a right threaded binary tree. In a two-waythreaded tree, also called a double-threaded tree, threads will appear in both the left and the right field ofthe node.
- While the left field will point to the in-order predecessor of the node, the right field will point to its successor. A two-way threaded binary tree is also called a fully threaded binary tree.

www.binils.com

Figure shows a binary tree without threading and its corresponding linked representation.

The in-order traversal of the tree is given as *8, 4, 9, 2, 5, 1, 10, 6, 11,3, 7, 12*

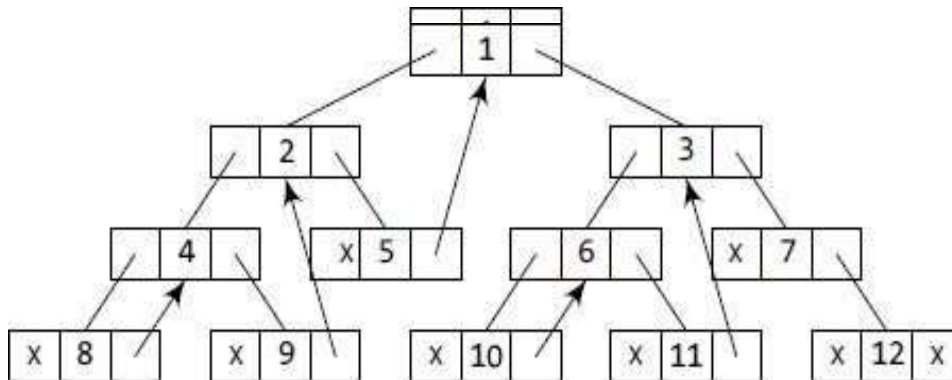


Download Binils Android App in Playstore

Download Photoplex App

One-way Threading

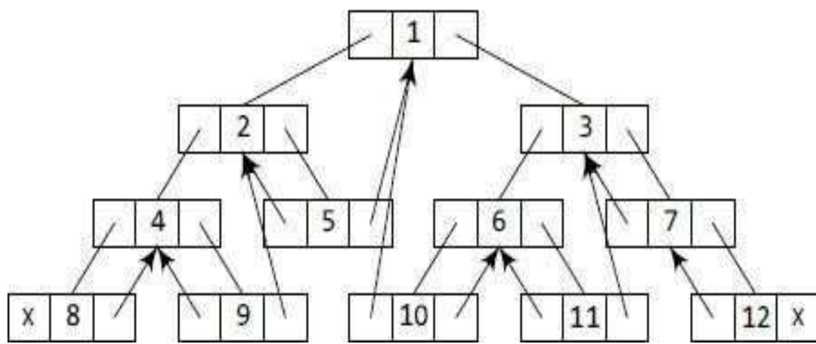
- Given Figure shows a binary tree with one-way threading and its corresponding linked representation.
- Node 5 containsa NULLpointer in its RIGHT field, so it willbe replaced to point to node1, which is its in-order successor. Similarly, the RIGHT field of node 8 will point to node 4, the RIGHT field of node 9 will point to node 2, the RIGHT field of node 10 will point to node 6, the RIGHT field of node 11 will point to node 3, and the RIGHTfieldofnode 12 willcontain NULLbecause ithasnoin-ordersuccessor.

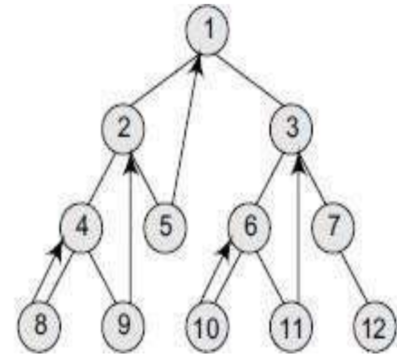


Two-way Threading

www.binils.com

- Given Figure shows a binary tree with two-way threading and its corresponding linked representation. Node 5 contains a NULL pointer in its LEFT field, so it will be replaced to point to node 2, which is its in-order predecessor. Similarly, the LEFT field of node 8 will contain NULL because it has no in-order predecessor, the LEFT field of node 7 will point to node 3, the LEFT field of node 9 will point to node 4, the LEFT field of node 10 will point to node 1, the LEFT field ofnode 11 willcontain 6, and the LEFT field of node 12 will point to node 7.





a) Linked representation of the binary tree with threading **(b) binary tree with two-way threading**

Now, let us look at the memory representation of a binary tree without threading, with one-way threading, and with two-waythreading. This is illustrated in given Fig.

ROOT = 3, AVAIL = 15

(a)

	LEFT	DATA	RIGHT
1	–1	8	–1
2	–1	10	–1
3	5	1	8
4			
5	9	2	14
6			
7			
8	20	3	11
9	1	4	12
10			
11	–1	7	18
12	–1	9	–1
13			
14	–1	5	–1
15			
16	–1	11	–1
17			
18	–1	12	–1
19			
20	2	6	16

ROOT = 3, AVAIL = 15

(b)

	LEFT	DATA	RIGHT
1	–1	8	9
2	–1	10	20
3	5	1	8
4			
5	9	2	14
6			
7			
8	20	3	11
9	1	4	12
10			
11	–1	7	18
12	–1	9	5
13			
14	–1	5	3
15			
16	–1	11	8
17			
18	–1	12	–1
19			
20	2	6	16

ROOT = 3, AVAIL = 15

(c)

	LEFT	DATA	RIGHT
1	–1	8	9
2	3	10	20
3	5	1	8
4			
5	9	2	14
6			
7			
8	20	3	11
9	1	4	12
10			
11	8	7	18
12	9	9	5
13			
14	5	5	3
15			
16	20	11	8
17			
18	11	12	–1
19			
20	2	6	16

Memory representation of binary trees: (a) without threading, (b) with one-way, and (c) two-way threading

Traversing a Threaded Binary Tree

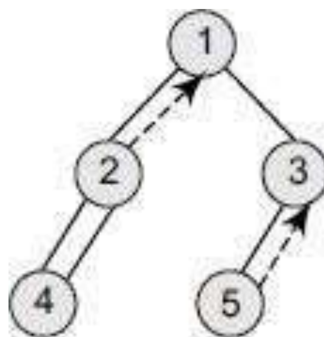
For every node, visit the left sub-tree first, provided if one exists and has not been visited earlier. Then the node (root) itself is followed by visiting its right sub-tree (if one exists). In case there is no right sub-tree, check for the threaded link and make the threaded node the current node in consideration. The algorithm for in-order traversal of a threaded binary tree is given in Fig.

Step 1: Check if the current node has a left child that has not been visited. If a left child exists that has not been visited, go to Step 2, else go to Step 3.
Step 2: Add the left child in the list of visited nodes. Make it as the current node and then go to Step 6.
Step 3: If the current node has a right child, go to Step 4 else go to Step 5.
Step 4: Make that right child as current node and go to Step 6.
Step 5: Print the node and if there is a threaded node make it the current node.
Step 6: If all the nodes have visited then END else go to Step 1.

www.binils.com

Algorithm for in-order traversal of a threaded binary tree

Let's consider the threaded binary tree given in Fig. and traverse it using the algorithm.



Threaded binary tree

1. Node 1 has a left child i.e., 2 which has not been visited. So, add 2 in the list of visited nodes, make it as the current node.

2. Node 2 has a left child i.e., 4 which has not been visited. So, add 4 in the list of visited nodes, make it as the current node.

3. Node 4 does not have any left or right child, so print 4 and check for its threaded link. It has a threaded link to node 2, so make node 2 the current node.

4. Node 2 has a left child which has already been visited. However, it does not have a right child. Now, print 2 and follow its threaded link to node 1. Make node 1 the current node.

5. Node 1 has a left child that has been already visited. So print 1. Node 1 has a right child 3 which has not yet been visited, so make it the current node.

6. Node 3 has a left child (node 5) which has not been visited, so make it the current node.

7. Node 5 does not have any left or right child. So print 5. However, it does have a threaded link which points to node 3. Make node 3 the current node.

8. Node 3 has a left child which has already been visited. So print 3.

9. Now there are no nodes left, so we end here. The sequence of nodes printed is—421 5 3.

www.binils.com

Advantages of Threaded Binary Tree

- It enables linear traversal of elements in the tree.
- Linear traversal eliminates the use of stacks which in turn consume a lot of memory space and computertime.
- It enables to find the parent of a given element without explicit use of parent pointers.
- Since nodes contain pointers to in-order predecessor and successor, the threaded tree enables forward and backward traversal of the nodes as given by in-order fashion.

Thus, we see the basic difference between a binary tree and a threaded binary tree is that in binary trees a node stores a NULL pointer if it has no child and so there is no way to traverse back.

Programming Example

1. Write a program to implement simple right in-threaded binary trees.

```
#include<stdio.h>

#include<conio.h>

struct tree

{

    int val;

    struct    tree    *right;

    struct tree *left;

    int thread;

};

struct tree *root = NULL;

struct tree* insert_node(struct tree *root, struct tree *ptr, struct tree *rt)

{

if(root == NULL)

{

root = ptr;

if(rt!= NULL)

{
```

www.binils.com

```
root–>right = rt;
root–>thread = 1;
}}
else if(ptr–>val< root–>val)
root–>left = insert_node(root–>left, ptr, root); else
if(root–>thread == 1)
{
    root–>right = insert_node(NULL, ptr, rt);
    root–>thread=0;
}
else
    root–>right = insert_node(root–>right, ptr, rt);
    return root;
}
struct tree* create_threaded_tree()
{
structtree*ptr;
int num;
printf("\n Enter the elements, press –1 to terminate ");
scanf("%d", &num);
while(num != –1)
```

www.binils.com

```
{
ptr=(struct tree*)malloc(sizeof(struct tree));
ptr>val =num;
ptr–>left=ptr–>right=NULL;
ptr–>thread =0;
root = insert_node(root, ptr, NULL);
printf(" \n Enter the next element ");
fflush(stdin);
scanf("%d", &num);
}
return root;
}
void inorder(struct tree *root)
{
struct tree *ptr=root, *prev;
do
{
while(ptr != NULL)
{
prev = ptr;
ptr = ptr–>left;}
```

www.binils.com

```
if(prev != NULL)
{
printf("%d",prev–>val);
ptr =prev–>right;
while(prev != NULL &&prev–>thread)
{
printf(" %d", ptr–>val);
prev = ptr;
ptr = ptr–>right;
}}
}while(ptr != NULL);
}
void main()
{
//    struct    tree    *root=NULL;
clrscr();
create_threaded_tree();
printf("\nThein–ordertraversalofthetreecanbe givenas:");
inorder(root);
getch();
}
```

www.binils.com

Output

Enter the elements, press –1 to terminate 5 Enter the

next element 8

Enter the next element 2

Enter the next element 3

Enter the nextelement 7 Enter the

nextelement–1

Thein–order traversal of the tree can be given as: 2 3 5 78

www.binils.com

INTRODUCTION

A tree is recursively defined as a set of one or more nodes where one node is designated as the root of the tree and all the remaining nodes can be partitioned into non-empty sets each of which is a sub-tree of the root.

Basic Terminology

Root node- The root node R is the topmost node in the tree. If R = NULL, then it means the tree

is empty. Sub-trees- If the root node R is not NULL, then the trees T1,T2,and T3 are called the

sub- trees of R.

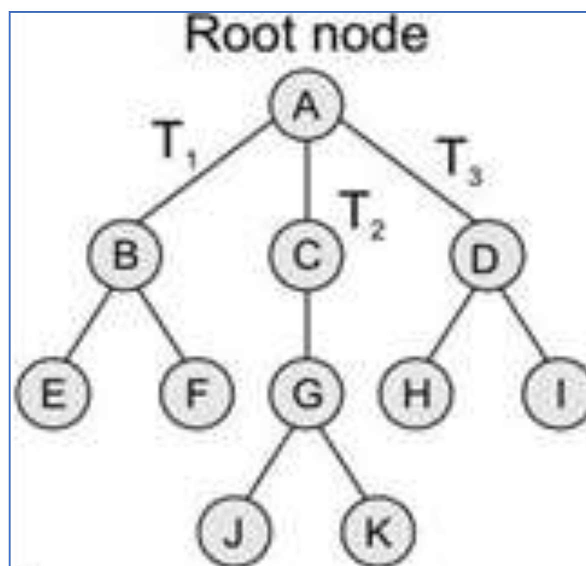
Leaf node - A node that has no children is called the leaf node or the terminal node.

Path - A sequence of consecutive edges is called a path. For example, in Fig., the path from the root node A to node I is given as: A, D, and I.

Ancestor node - An ancestor of a node is any predecessor node on the path from root to that node. The root node does not have any ancestors. In the tree given in Fig., nodes A, C, and G are the ancestors of node K.

www.binils.com

Descendant node- A descendant node is any successor node on any path from the node to a leaf node. Leaf nodes do not have any descendants. In the tree given in Fig., nodes C, G, J, and K are the descendants of node A.



TYPES OF TREES

Trees are of following 6 types:

- General trees
- Forests
- Binary trees
- Binary search trees
- Expression trees
- Tournament trees

General Trees

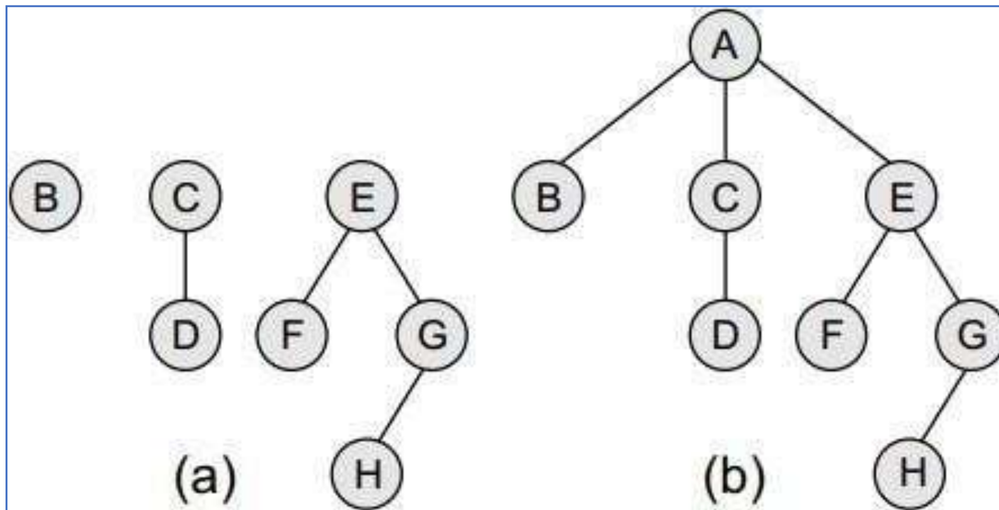
General trees are data structures that store elements hierarchically. The top node of a tree is the root node and each node, except the root, has a parent.

- A node in a general tree (except the leaf nodes) may have zero or more sub-trees.
- General trees which have 3 sub-trees per node are called ternary trees. 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.

www.binils.com

Forests

- Forest is a disjoint union of trees. A set of disjoint trees (or forests) is obtained by deleting the root and the edges connecting the root node to nodes at level 1.
- Every node of a tree is the root of some sub-tree. Therefore, all the sub-trees immediately below a node form a forest. We can convert a forest into a tree by adding a single node as the root node of the tree. For example, below Fig (a) shows a forest and Fig.(b) shows the corresponding tree. Similarly, we can convert a general tree into a forest by deleting the root node of the tree.



www.binils.com

TRAVERSING A BINARY TREE

Traversing a binary tree is the process of visiting each node in the tree exactly once in a systematic way. Unlike linear data structures in which the elements are traversed sequentially, tree is a non-linear data structure in which the elements can be traversed in many different ways.

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 by:

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

www.binils.com

```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2:             Write TREE -> DATA
Step 3:             PREORDER(TREE -> LEFT)
Step 4:             PREORDER(TREE -> RIGHT)
        [END OF LOOP]
Step 5: END
```

NOTE: Pre-order traversal is also called as depth-first traversal or NLR traversal algorithm (Node-Left-Right).

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 by:

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

```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2:            INORDER(TREE -> LEFT)
Step 3:            Write TREE -> DATA
Step 4:            INORDER(TREE -> RIGHT)
        [END OF LOOP]
Step 5: END
```

NOTE: In-order traversal is also called as symmetric traversal (or) LNR traversal algorithm (Left-Node-Right).

Post-order Traversal

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

1. Traversing the left sub-tree,

2. Traversing the right sub-tree, and finally

3. Visiting the root node.

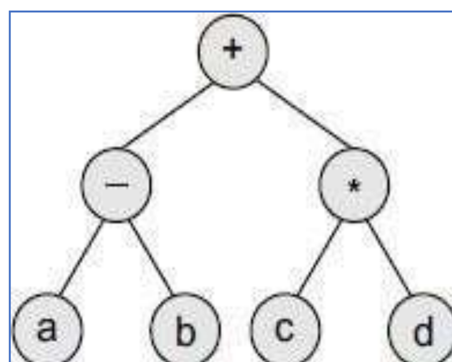
www.binils.com

```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2:            POSTORDER(TREE -> LEFT)
Step 3:            POSTORDER(TREE -> RIGHT)
Step 4:            Write TREE -> DATA
        [END OF LOOP]
Step 5: END
```

NOTE: Post-order algorithm is also known as the LRN traversal algorithm (Left-

Right- Node) Example 1: Find the In-order, Pre-order and post-order traversal of

given tree



Solution

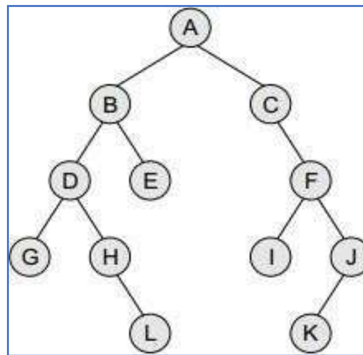
Pre-order Traversal : + – a b * c

d In-order Traversal : a-b+c*d

Post-order Traversal : ab-

cd*+

Example 2: Find the In-order, Pre-order and post-order traversal of given tree



www.binils.com

Solution

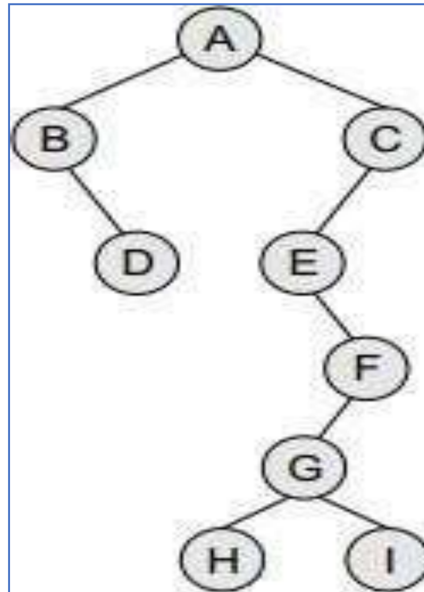
Pre-order Traversal : A, B, D, G, H, L, E, C, F, I, J,

and K In-order Traversal : G, D, H, L, B, E, A, C, I,

F, K, and J Post-order Traversal : G, L, H, D, E, B,

I, K, J, F, C, and A

Example 3: Find the In-order, Pre-order and post-order traversal of given tree



Solution

Pre-order Traversal : A, B, D, C, D, E, F, G, H,

and I In-order Traversal : B, D, A, E, H, G, I, F,

and C

www.binils.com

Post- order Traversal : D, B, H, I, G, F, E, C, and A

Level-order Traversal

In level-order traversal, all the nodes at a level are accessed before going to the next level. This algorithm is also called as the breadth-first traversal algorithm. Consider the trees given in Fig. and note the level order of these trees.