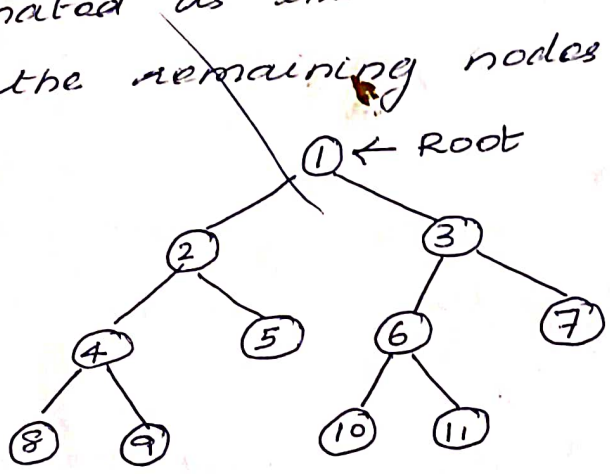
NON-LINEAR DATA STRUCTURES

Tree ADT – tree traversals – Binary Tree ADT Expression trees – applications of trees Binary search Tree ADT – Threaded Binary Tree AVL trees – B-tree – B+ tree – Heap – Applications of heap

Tree ADT:

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 form subtrees



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

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

Eg: 8, 9, 5, 10, 11, 7.

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

Eg: Degree (6) = 2

Siblings: All the nodes that share same parent are called siblings.

Eg: (8,9) (10,11)

Path: A sequence of consecutive edges

Eg: Path (1,8) = 1,2,4 and 8

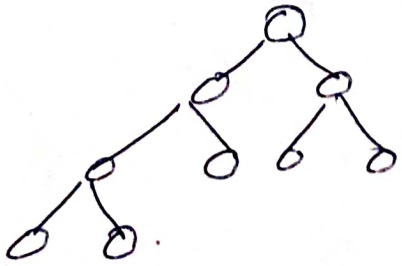
Depth: The depth of a node N is the length of the path from the root to the node N.

Eg: Depth (5) = 3
Depth (9) = 3

Depth of a root node is zero.

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

Complete binary tree: A complete binary tree is a binary tree, in which all the levels are completely filled from left except possibly the last.



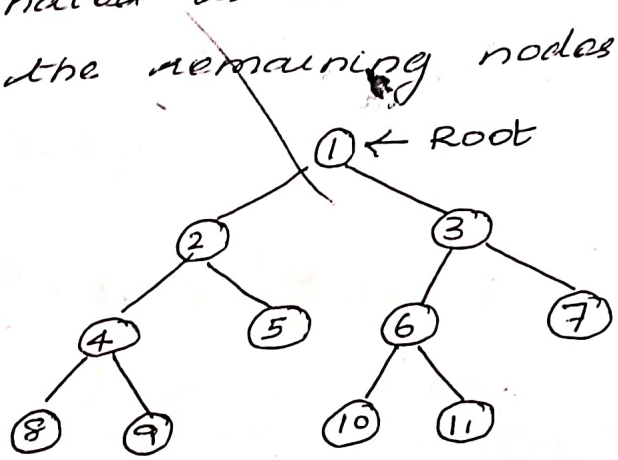
Height: The height of a node N is the length of the path from the node 'N' to root.

NON LINEAR DATA STRUCTURES

Tree ADT – tree traversals – Binary Tree ADT Expression trees – applications of trees Binary search Tree ADT – Threaded Binary Tree AVL trees – B–tree – B+ tree – Heap – Application of heap

Tree ADT:

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 form subtrees



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

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

Eg: 8, 9, 5, 10, 11, 7.

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

Eg: Degree (6) = 2

Siblings: All the nodes that share same parent are called siblings

Eg: (8,9) (10,11)

Path: A sequence of consecutive edges

Eg: Path (1,8) = 1,2,4 and 8

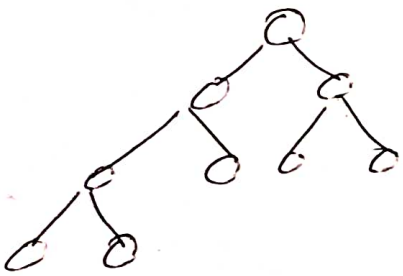
Depth: The depth of a node N is the length of the path from the root to the node N

Eg: Depth (5) = 3
Depth (9) = 3

Depth of a root node is zero.

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

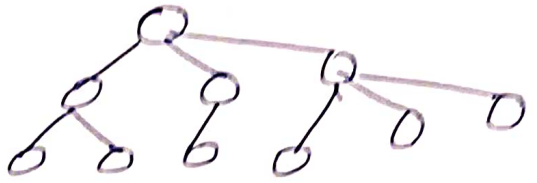
Complete binary tree: A complete binary tree is a binary tree, which all the levels are completely filled from left except possibly the last



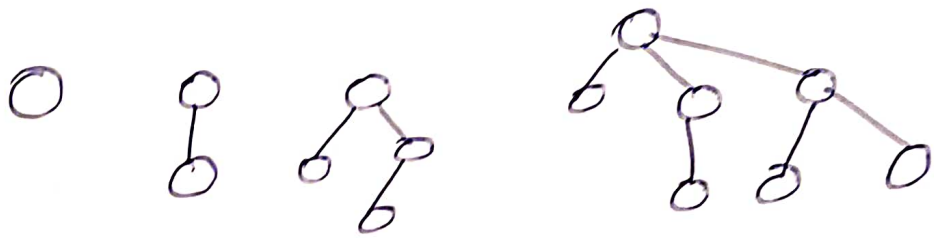
Height: The height of a node N is the length of the path from the node 'N' to root.

Types of trees:

- General trees: General trees store elements hierarchically. A node in a general tree may have zero or more subtrees.

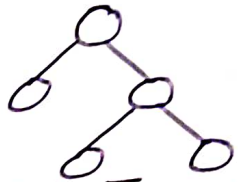


- Forests: A forest is a disjoint union of trees. A forest is also defined as an ordered set of zero or more general trees.

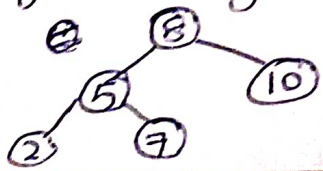


A forest.

- Binary Trees: Binary tree is a tree with not more than 2 children. Every node can have 0, 1, or at most 2 children.



- Binary search Tree (BST): Binary search Tree is a binary tree with a condition that left child is smaller than the root and right is greater than the ro



Tournament trees: In a tournament tree, each external node represents a player and each internal node represents the winner of the match played between the players represented by its children.

TREE TRAVERSALS:

- Traversal is the process of visiting each node in the tree exactly once in a systematic way.
- Types traversals
  - ⇒ Inorder traversal
  - ⇒ Preorder traversal.
  - ⇒ Post order traversal.
  - ⇒ Level-order Traversal

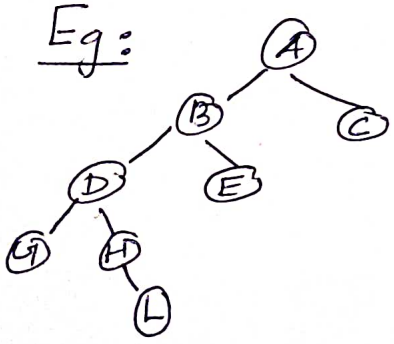
Inorder traversal:

To traverse a non-empty binary tree in in-order the following operations are performed recursively at each node

Algorithm:

1. Traverse the left subtree
2. Visit the root node
3. Traversing the right subtree

Eg:



Inorder traversal:

GDHLBEAC

In-order traversal is also called as symmetric traversal. In-order algorithm is also known as the LNR traversal algorithm (Left-Node-Right).

Routine:

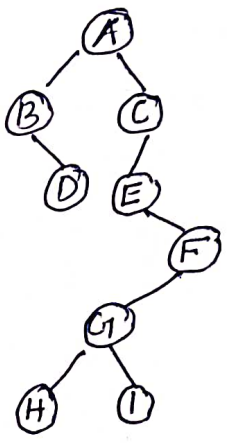
```
void    Inorder (Tree T)
 {
    if(T != NULL)
      {
        Inorder (T→Left);
        Write T→Data;
        Inorder (T→Right);
      }
 }
```

Preorder Traversal:

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

Algorithm:

1. Visit the root node
2. Traverse the left sub-tree
3. Traverse the right sub tree



A B D C E F G H I
(pre order)

Preorder algorithm is also known as the NLR traversal algorithm (Node - Left - Right). Preorder traversal algorithms are used to extract a pr notation from an expression tree.

Routine

```
void Preorder ( Tree T)
{
    if (T != NULL)
    { Write Node T→Data;
      Preorder (T→Left);
      Preorder ( T→Right);
    }
}.
```

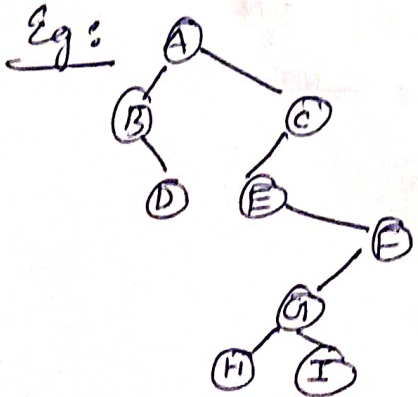
Post order Traversal:

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

Algorithm:

1. Traverse the left sub-tree
2. Traverse the right subtree
3. Visit the root node.

Eg:



D B H I G F E C A

Post-order algorithm is also known as LRN traversal algorithm (Left-Right-Node) Post order traversals are used to extract postfix notation from an expression tree.

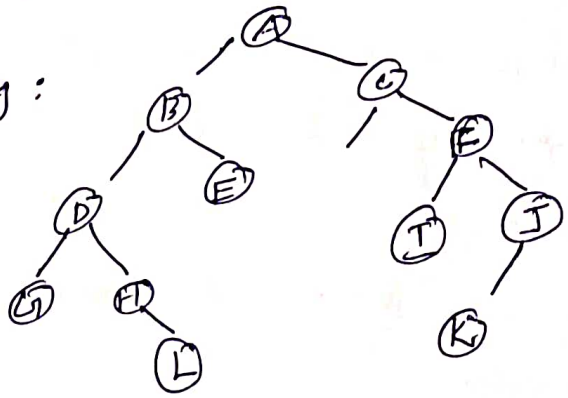
Routine:

```
Void Postorder (Tree T)
{
  if (T != NULL)
    {
      Postorder (T->Left);
      PostOrder (T->Right);
      Write T->Data;
    }
}.
```

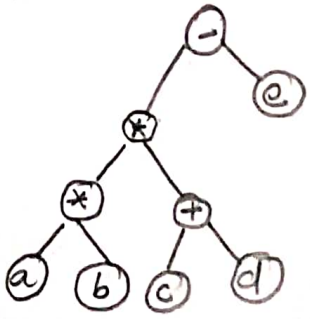
Level order Traversal:

In level order traversal, all the nodes at same level are accessed before going to the next level. This algorithm is also called as the breadth-first travel algorithm.

Eg:

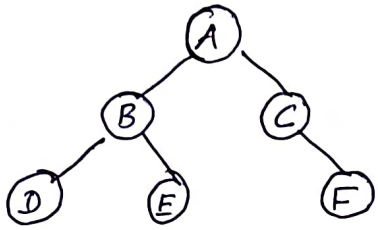


A B C D E F G H I J K L
(Traversal)



BINARY TREE ADT:

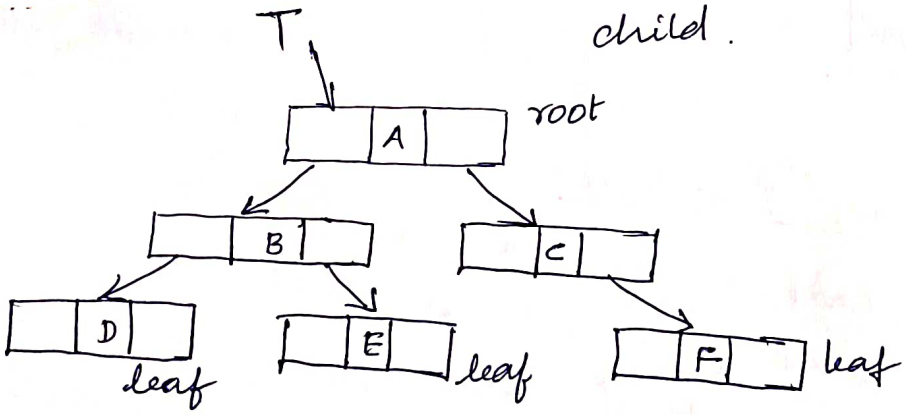
Binary tree is a tree in which no node can have more than 2 children. Every node can have either 0, 1 or 2 children.



Eg: Binary Tree

A binary tree of height h h
at least 'h' nodes and at
$2^h - 1$ nodes.
A binary tree of 'n' nodes
exactly n-1 edges
The height of a binary t
with 'n' nodes is at least l
and atmost n.

It is also defined as collection of nodes and top most node is root node. Every node contains data element, a left pointer which points to the child, and a right po which points to the r child.



- Root node is pointed by a pointer 'T'.
- Every node in a tree is connected by a direc edge from exactly one other node ie parent
- A node can have 0, 1 or 2 children.
- Nodes with no children are called leave external nodes.
- Nodes with same parent are called siblings

Node declaration:

```
struct Node
{
    ElementType Element;
    Tree        Left;
    Tree        Right;
};
```

- The depth of a node 'x' is the number of edges from root to the node 'x'.
- The height of a node 'x' is the number of edges from node 'x' to the deepest leaf.
- Height of a tree is the height of the root.
- Complete binary tree is a binary tree which is completely filled from left to right except the last level.

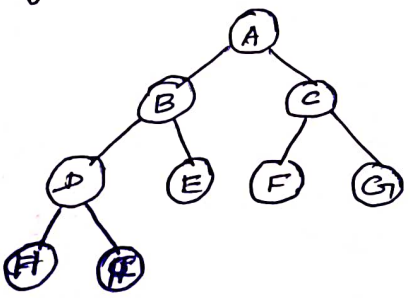


Fig: Complete binary tree.

Full binary tree is a tree in which every node other than the leaves has 2 children.

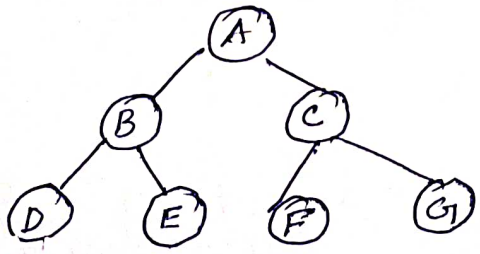


Fig: Full binary tree

Representation of Binary Trees:

- Linked list representation.
- Sequential or Array representation.

Linked list representation:

In this, every node will have three parts

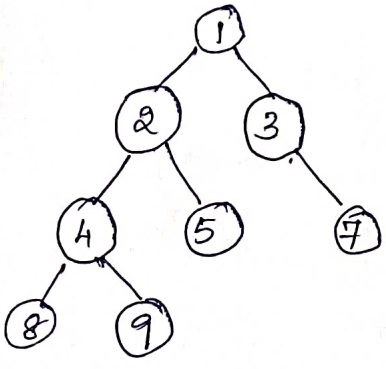
Data → Data to be stored.

Left → Pointer to the left sub tree

Right → Pointer to the right subtree.

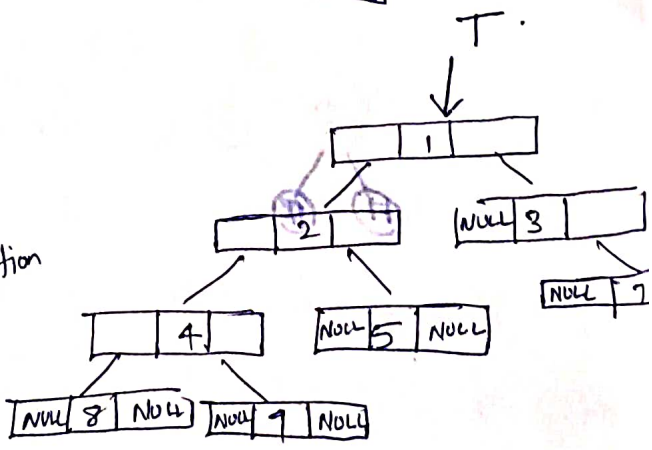
Node declaration:

```
struct Node
{
    ElementType Element;
    struct Node *Left;
    struct Node *Right;
};
```



A Binary Tree

linked list representation



Linked List represen

Root node is pointed by a pointer T.

If 'T' is NULL, then tree is empty.

Sequential representation / Array representation

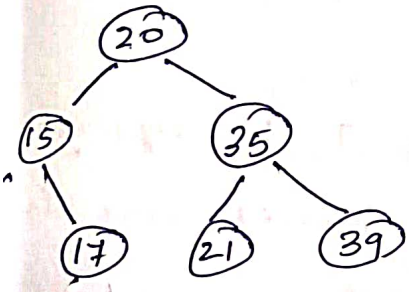
A single dimensional array is used for sequential representation. This is the simplest technique for memory representation.

- Root of the tree is stored in the 1st location.
- Left subtree of x

  Left(x) = 2(i), i → position of x

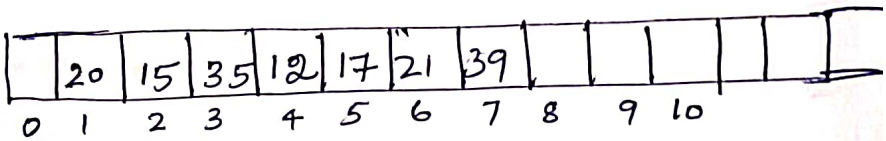
- Right subtree of x

  Right(x) = 2i + 1,



Binary Tree

Array / sequential representation



x = 20, i = 1

Left(x) = Left(20) = 2i = 2(1) = 2

Right(x) = Right(20) = 2i + 1 = 3

x = 15, i = 2

Left(15) = 2i = 2 * 2 = 4

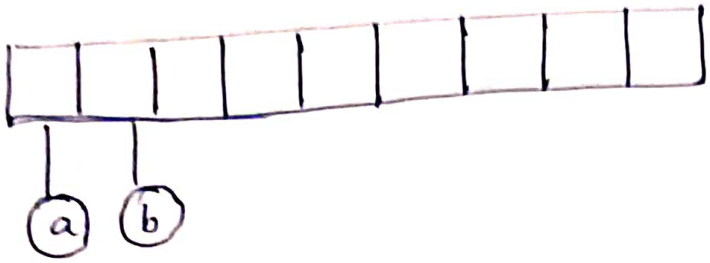
Right(15) = 2i + 1 = 4 + 1 = 5

x = 35, i = 3

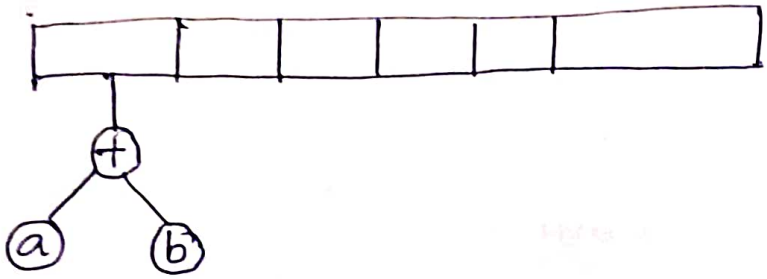
Left(35) = 2 * 3 = 6

Right(35) = 2 * 3 + 1 = 7

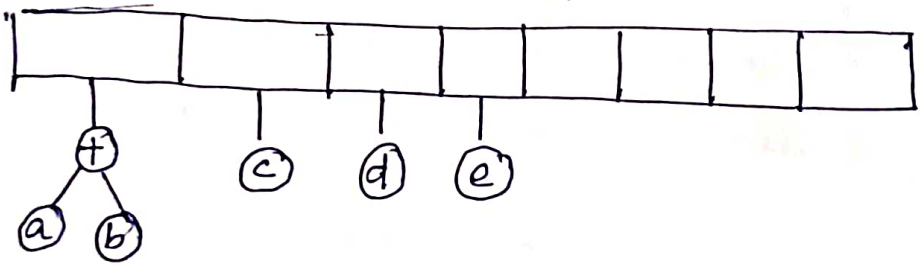
3. Next input is b, an operand, create a new node



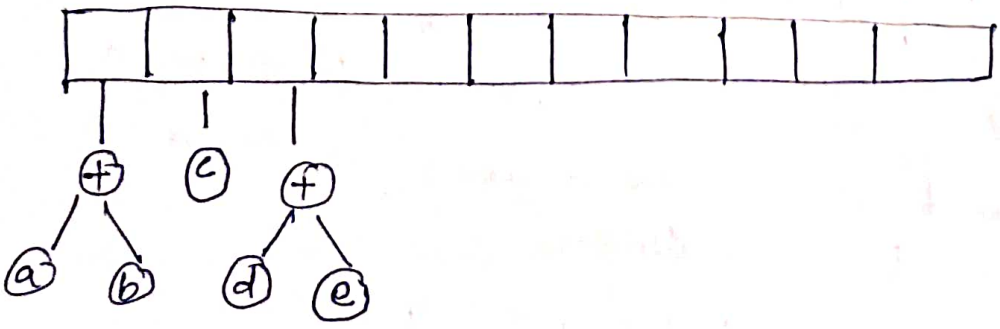
4. Next input is +, an operator, pop top 2 and create a new tree



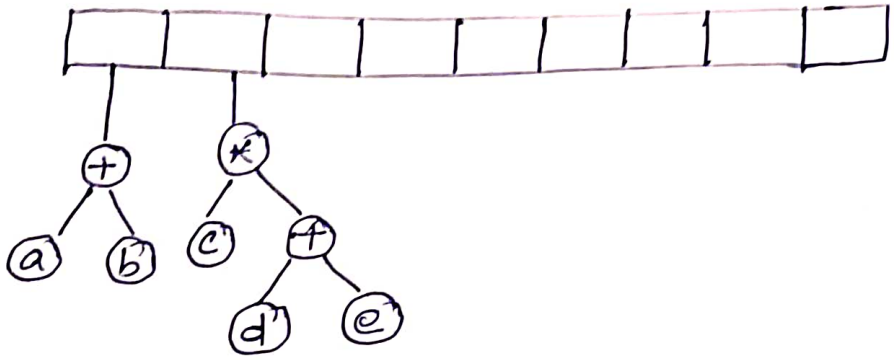
5. Next 3 inputs are operands ie b, d, e, create new nodes and push it



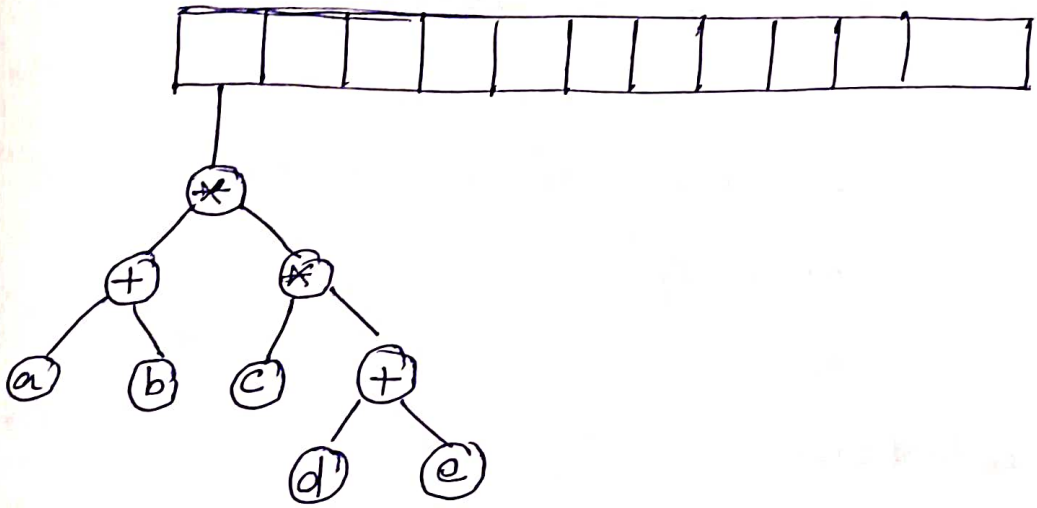
6. Next input is +, pop top 2 and creat a new tree



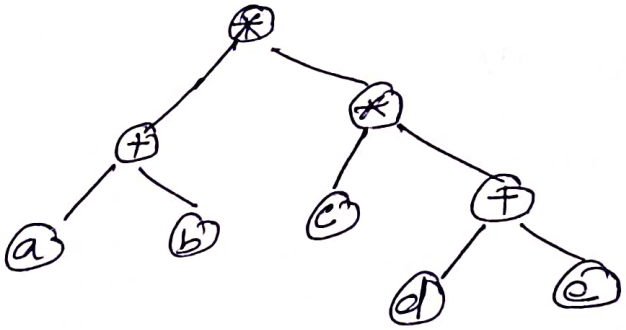
7. Next input is an operator '*', pop 2, and create new tree.



8. Next input is an operator *, pop 2 elements, and create new subtree.



Final expression tree is



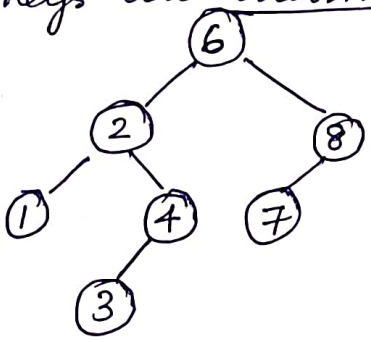
Applications of trees:

- Trees are used to store simple and complex data. Simple means an integer and character values. Complex data means a structure or a reco
- Trees are often used for implementing oth types of data structures like hash tables, sets and maps.
- A self-balancing tree, Red-black tree is used in kernal scheduling, to preempt massively multiprocessor computer operating system use
- B-trees are prominently used to store tre structures on disc. They are used to in a large number of records.
- B-trees are also used for secondary in in databases, where the index facilit a select operation to answer some ran criteria.
- Trees are an important data structu 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.

BINARY SEARCH TREE ADT: (BST)

Binary search Tree is a binary tree with a condition that the key values of left subtree is smaller than the root node and the key values of right subtree is greater than the root node.

- Thus, every node is assigned with a key value
- All the keys are distinct.



Binary search Tree

The root node is 6. The left subtree of root node consists of nodes 1, 2, 3, and 4. The right sub tree consists of 7, and 8.

For the left subtree root node is 2, The left has 1 and right has 3 and 4.

Thus recursively it applies to all the nodes.

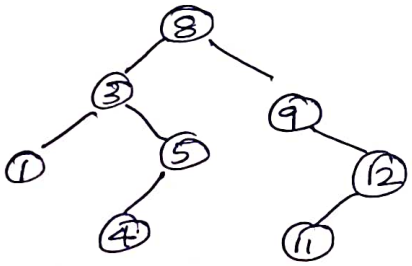
- Since the nodes in the binary search tree are ordered, the time needed to search an element in the tree is reduced.
- Whenever we search for an element, do not need to traverse the entire tree
- Binary search trees are widely used in dictionary problems.

Operations on BST:

1. Insertion
2. Deletion
3. Find
4. Find Min
5. Fin Max.

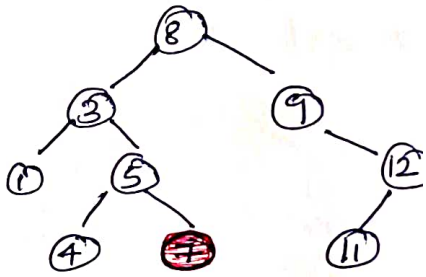
(i) Insertion:

We start at the root and recursively go d the tree for appropriate location and inserts node. If the element to be inserted is alree in the tree, no need to insert.



Before insertion.

Insert 7



After insertion.

Node declaration:

```
Struct Node
{
    ElementType Element;
    Struct Node * Left;
    Struct Node * Right;
};
```

Routine for insertion:

```
SearchTree  Insert ( ElementType X , SearchTree T )
{
   if ( T == NULL)
      {
        T = malloc (sizeof (struct Node));
        if (T==NULL)
            FatalError ("out of space");
        else
            {
              T→Element = X;
              T→ Left  = T→Right = NULL;
            }
      }
   else
      {
        if ( X< T→Element )
              T→Left = Insert ( X, T→Left);
        else  if ( X > T→Element )
                    T→Right = Insert ( X, T→Right);
              else
                    return T;
      }
```

Deletion:

Unlike insertion, deletion ~~case~~ has different ceases

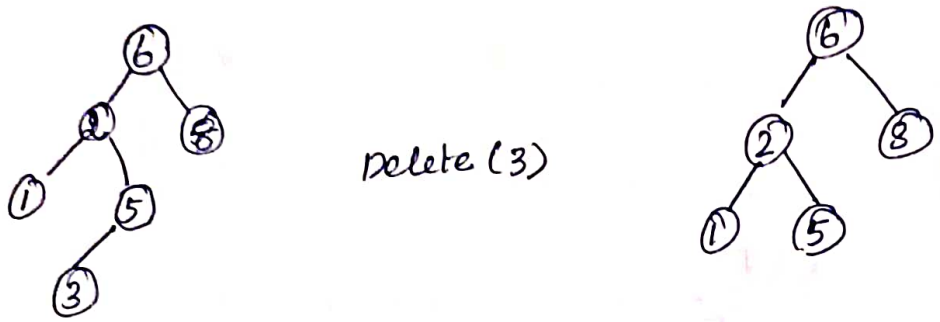
case 1: Deleting a leaf node

case 2: Deleting a node with one child

case 3: Deleting a node with 2 children.

Case 1 : Deleting a leaf node

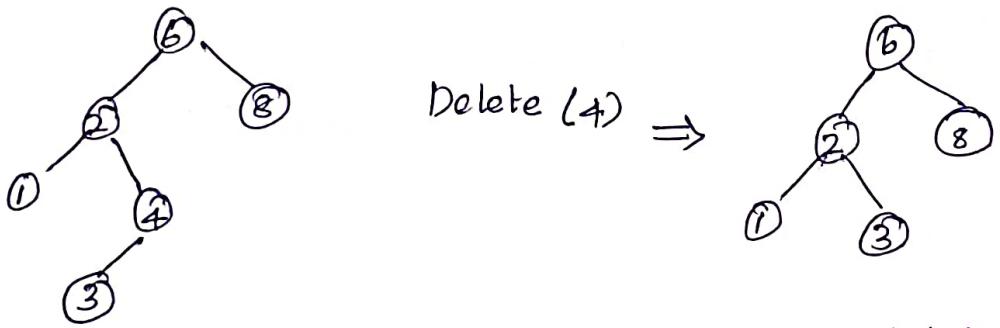
If a node is a leaf, it can be deleted immediately.



Before Deletion. After deletion.

Case 2 : Deleting a node with one child.

If the node has one child, the node can be de after its parent adjusts a pointer to bypass node

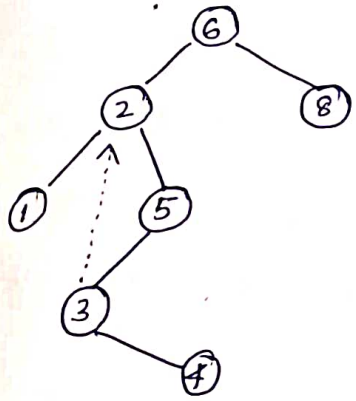


After deletion.

Before deletion.

case 3: Deleting a node with 2 children:

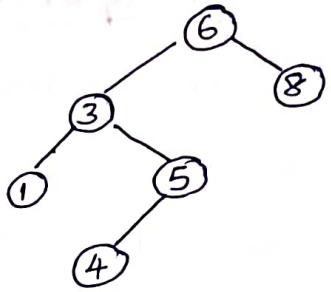
Replace the node to be deleted with the smallest data in the right subtree and delete the node.



Before Deletion

Delete (2)

Find the min at the right subtree i.e 3 replace (2) with (3)



After deletion.

Routine for deletion:

```
SearchTree Delete (ElementType X, SearchTree T)
{
  Position TmpCell;
  if (T==NULL)
      Error ("Element not found");
  else
     if (X< T->Element)
          T->Left = Delete (X, T->Left)
  else if (X > T->Element)
          T->Right = Delete ( X, T->Right)
       else if ( T->Left && T->Right) // Two children
            {
              Tmpcell= Find Min (T->Right);
              T->Element = TmpCell -> Element ;
              T->Right = Delete ( T->Element ,T->Right
            }
       else
            {
              TmpCell =T;
              if ( T->Left ==NULL)
                    T= T->Right;
              else if (T->Right == NULL)
  return T;} free ( TmpCell); T->Left;
```

(iii) Find operation

Find operation start comparing with root node and goes towards down until it finds the element. Once it found, returns the address of the node.

Routine:

```
Position Find ( ElementType X, SearchTree T)
{
if (T==NULL)
      return NULL;
if ( X< T→Element )
      return Find ( X,T→Left);
else if ( X>T→Element )
      return Find ( X,T→Right);
else
      return T;
}
```

(iv) Find Min operation:

Returns the minimum element in the tree

```
Position FindMin ( SearchTree T)
{
  if ( T==NULL)
      return NULL;
  else if ( T→Left ==NULL)
      return T;
  else return FindMin ( T→Left);
}
```

(v) Find Max Operation

Returns the maxi element in the

```
Position FindMax (SearchTree
{
  if (T==NULL)
      return NULL;
  else if (T→Right==N
      return T;
  else
      return FindMax
}
```

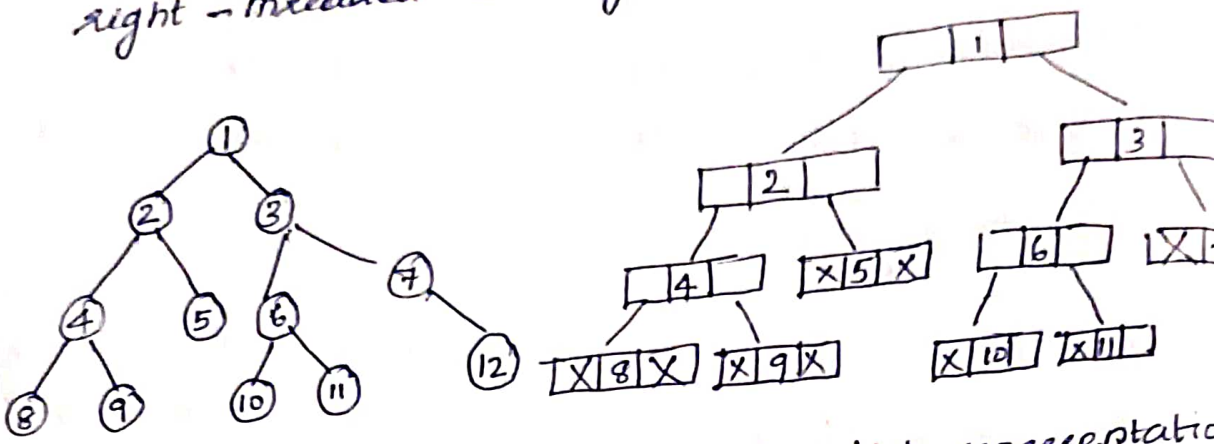
THREADED BINARY TREES:

- A threaded binary tree is a binary tree, with pointers in the leaves pointing the predecessor instead of being NULL.
- In linked list representation of a binary tree, left and right pointers will be NULL.. This space is wasted. To efficiently use this space, NULL entries are replaced with some values.
- These pointers are called as threads and binary tree containing threads are called threaded binary trees.
- There are 2 ways of threading
  - One way threading
  - Two way threading.

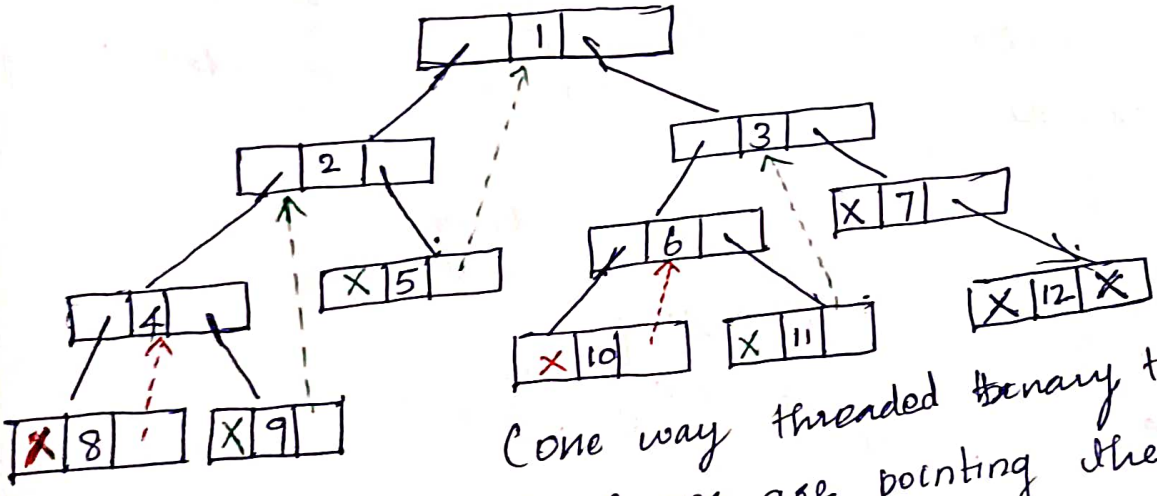
One way threading:

- A thread will appear in the right field or left field of the node
- 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
- Such a one way threaded tree is called a left-threaded binary tree.
- If the thread appears in the right field, then it will point to the in-order successor

- Such a one~~ne~~ way threaded tree is called a right-threaded binary tree.



Linked list representatio
(without threading



(One way threaded ~~B~~inary trees)

- Right pointers ~~of~~ leaves are pointing the inor successor
- Left pointers are made as NULL
- Inorder successors are calculated from inorder traversal of the tree.

Inorder traversal: 8 4 9 2 5 1 10 6 11

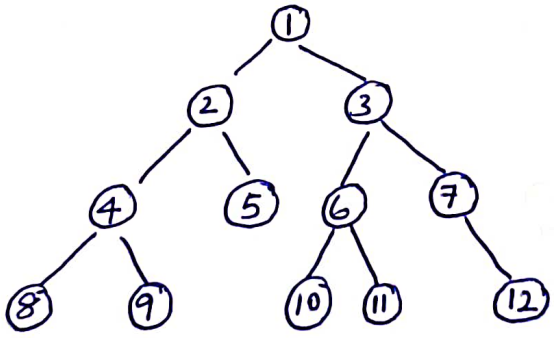
- Element followed by the leaves ~~are~~ is success

inorder successor of 8 is 4, 9 is 2, 1
11 is 3,

12 has no successor

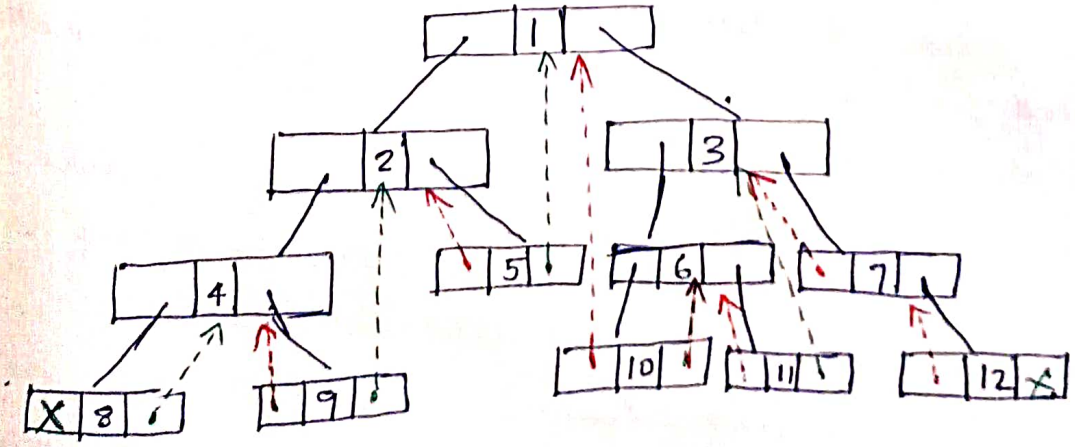
Two way threading:

- In a two way threaded binary tree, threads will appear in both the left and right field of the node
- It is also called as double threaded tree.
- Left field will point the inorder predecessor of the node
- Right field will point to its successor
- It is also called as fully threaded binary tree



Binary tree

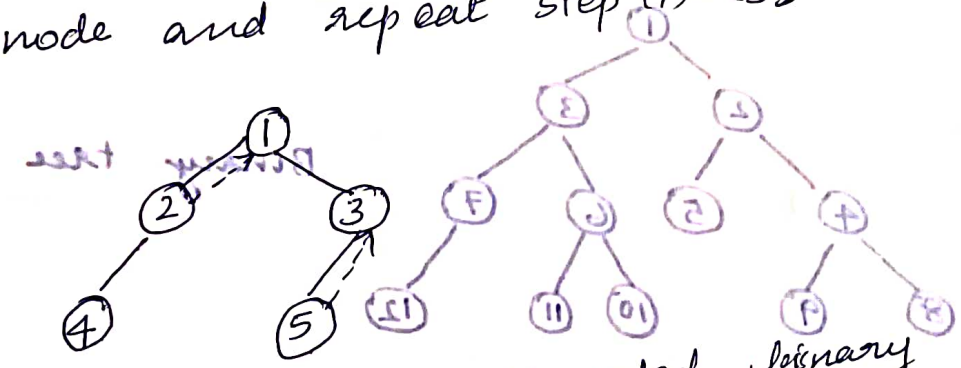
Inorder traversal : 8 4 9 2 5 1 10 6 11 3 7 12



Traversing a threaded Binary Tree:

Algorithm:

1. For every node, if there is a left subtree, mark it as visited, if not visited
2. Then consider the visited node as the current node and check for its left, and if there is no left and right ~~unvisited~~, print the current node.
3. Print the root node and check for its right, if no right follow the threaded link and go back to the previously visited node and repeat step (1)-(3)



Let us consider the threaded binary tree in the above fig and traverse it using the algorithm

1. Node 1 has a left child ie 2 which has not been visited, so add 2 in the list of unvisited, make as the current node.
2. Node 2 has a left child ie 4 which has not been visited. so add 4 in the list of visited nodes and make it as current node.
3. Node 4 does not have any left or right child, so print 4 and check for its thread link. It has a threaded link to node 2, so make 2 the current. Thus repeat for all nodes

Advantages of Threaded Binary tree:

1. It enables linear traversal of elements in the tree.
2. Linear traversal eliminates the use of stacks which in turn consume a lot of memory space and computer time.
3. It enables to find the parent of a given element without explicit use of parent pointers.
4. Since nodes contain pointers to inorder predecessor and successor, the threaded tree enables forward and backward traversal.

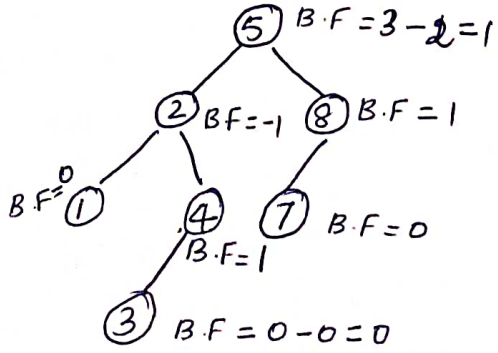
AVL TREES: (Adelson Velsky Landis)

An AVL tree is a binary search tree in which every node has a balance factor of 0, -1, or 1

> Balance factor = Height of left subtree − Height of right subtree

A node with any other balance factor is considered to be unbalanced and requires rebalancing of tree.

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



An AVL tree.

- AVL tree should be maintained balanced.
- During insertion or deletion violation may occurs. To rebalance the tree rotation is performed on tree.

ROTATION:

- Rotation is the process of changing the positions of the node, inorder to rebalance the tree.
- Violation may occur in following cases
  1. An insertion into the left subtree of the left child of α
  2. An insertion into the right subtree of the left child of α.
  3. An insertion into the left subtree of the right child of α.
  4. An insertion into the right subtree of the right child of α.

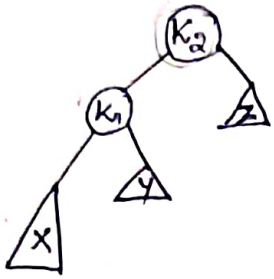
$\alpha \rightarrow$ Violated Node

Types of rotations.

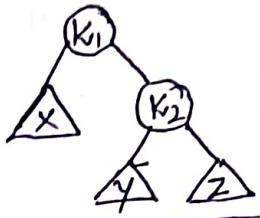
- Single rotation.
  - → Left Left rotation. (LL - Rotation)
  - → Right Right rotation (RR - Rotation)
- Double Rotation
  - → Left – Right Rotation (LR - Rotation)
  - → Right - Left Rotation (RL - Rotation)

LL Rotation:

When the node is inserted at the left subtree of left child of violated node α, LL rotation is applied

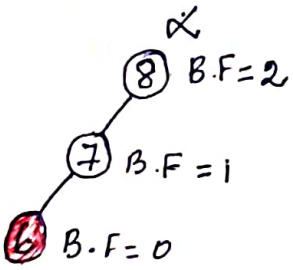


LL Rotation $\Rightarrow$

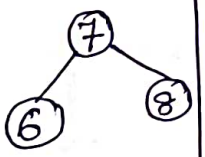



```
Position SingleRotateWithLeft
                        Position
{
  Position K1;
  K1 = K2→Left;
  K1→Right = K2;
  K2→Left = K1→Right;
K2→Height = Max(Height (
                    Height (K2
K1→Height = Max(Height (K1→
                    K2→Left
  return K1;
}.
```

Eg:1:

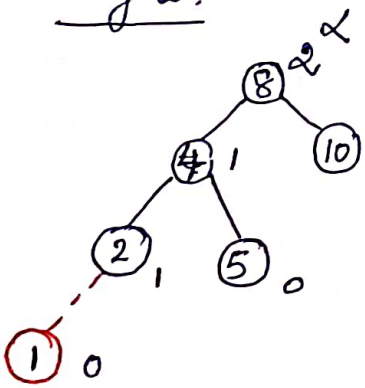


LL(8) $\Longrightarrow$

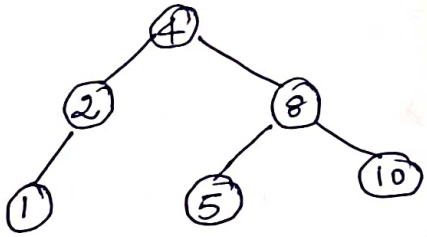


After Inserting (6) violation occurs at (8), which is α, perform LL-rotation on (8)

Eg 2:



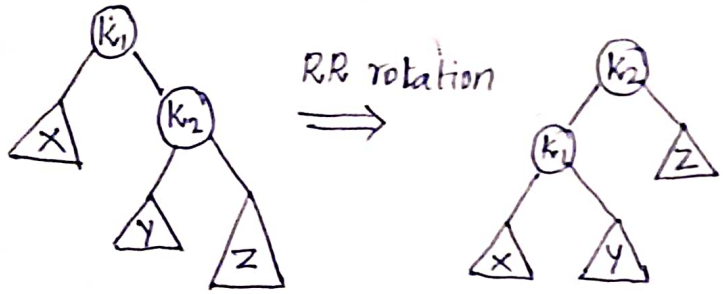
LL(8) $\Longrightarrow$



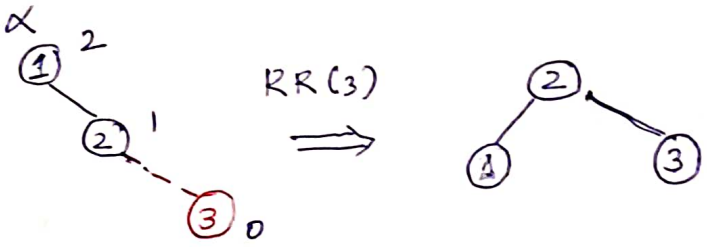
Inserting (1), (2) is violated.

RR Rotation:

When the node is inserted at the right subtree of right child of violated node α RR rotation is applied.

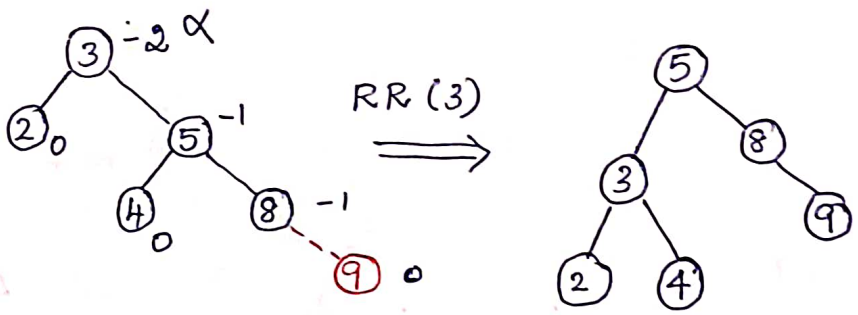


Eg:



Inserting (3)

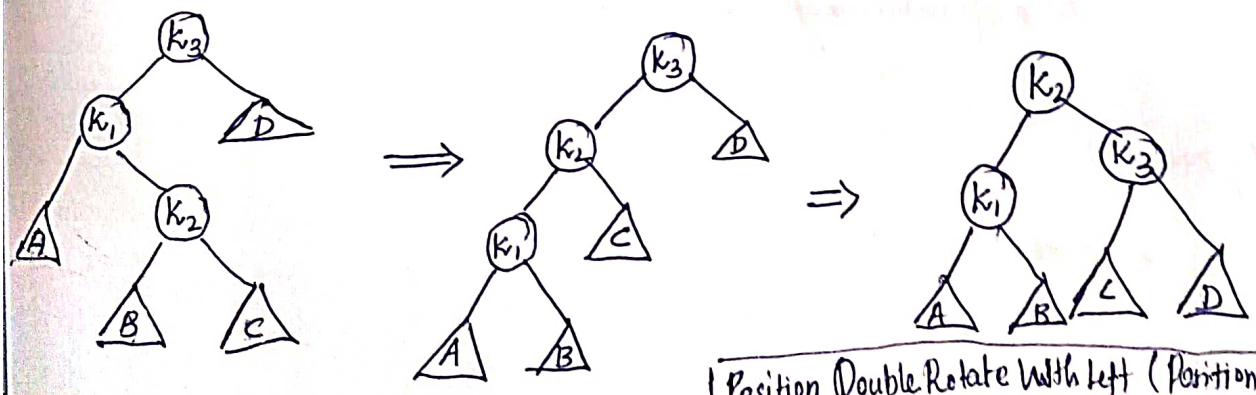
Eg 2:



Inserting (9)

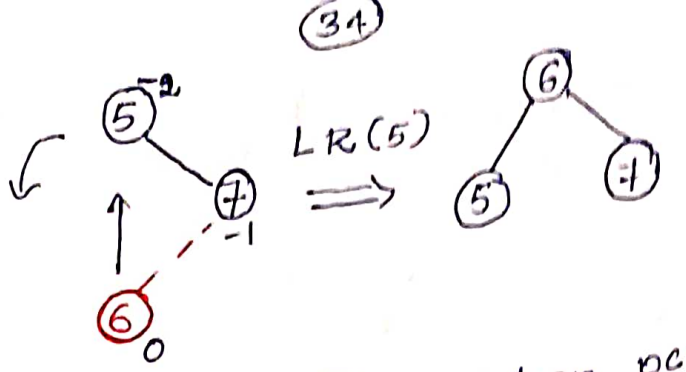
LR Double Rotation:

- When a node is inserted at the right subtree of left child of violated node α, LR rotation is applied.




```
Position DoubleRotate WithLeft ( Position K3)
{
  K3->Left = SingleRotateWithRight (K3->Left);
  return SingleRotateWithLeft(K3);
}
```

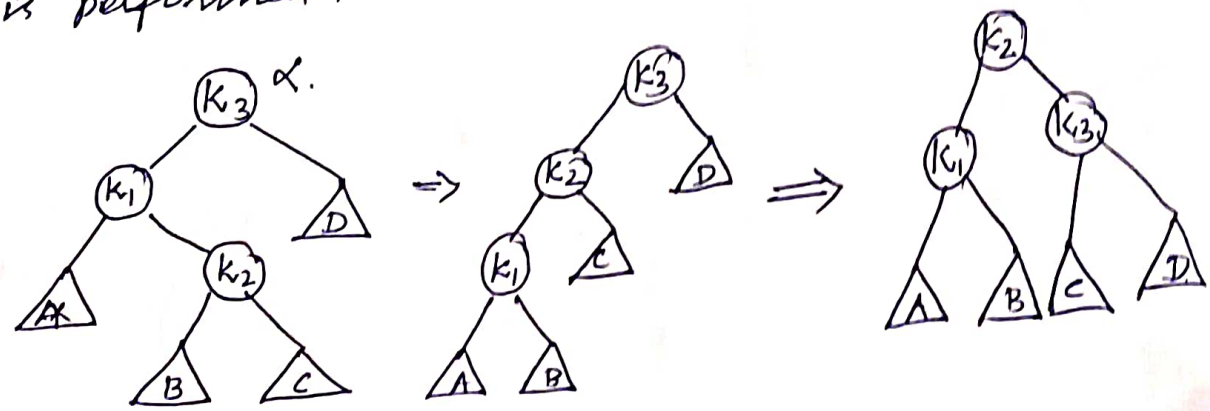
Eg:



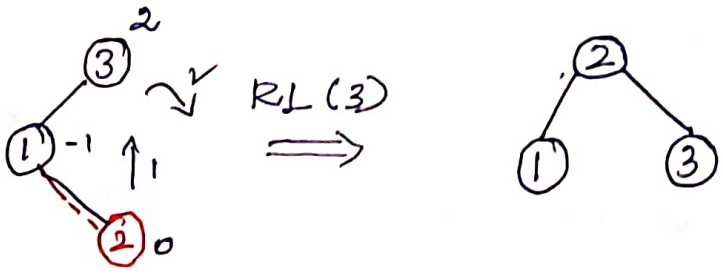
After inserting (6), violation occcurs at 5.

RL Rotation:

When a node is inserted at the left subtree of right child of violated node α, RL rotation is performed.



Eg 1:



After Inserting (2)

Operations:

- Insertion operation.
- Deletion operation.
- FindMin. operation
- FindMax. operation
- Find operation

Insertion:

Insertion is performed similar as binary search tree. New node is compared with the root node, if key value of new node is smaller than root node, then inserted at the left and if greater than root, then inserted at the right.

After every insertion balance condition is checked for every node, if violation occurs rotation is applied.

Routine:

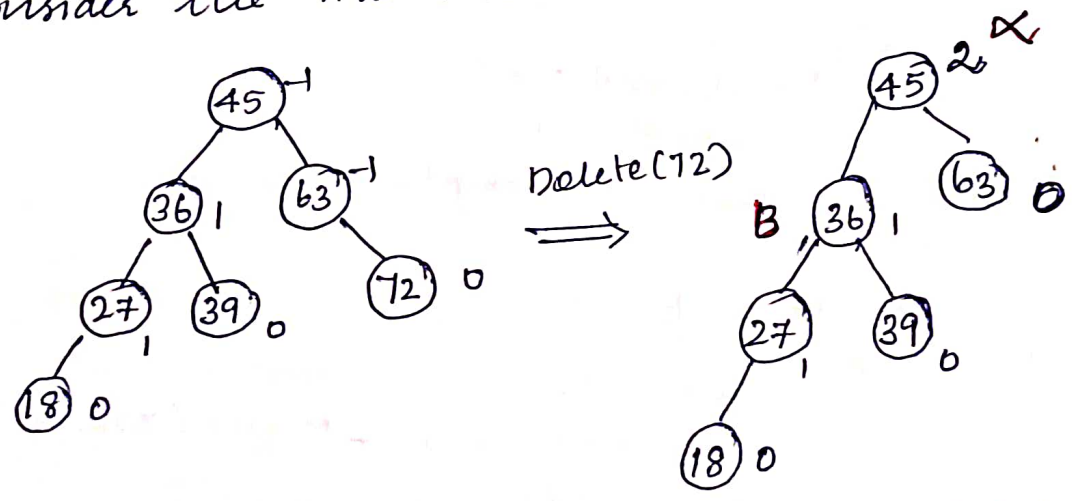
```
AVLTree Insert ( ElementType X, AVLTree T)
{
  if (T == NULL)
    {
      T= malloc(sizeof( struct AVLNode);
      if (T==NULL)
          FatalError ("Out of space");
      else {
          T→Element = X;
          T→ Height = 0;
          T→ Left  =T→Right = NULL;
        }
  else if ( X< T→ Element )
        {
          T→ Left = Insert ( X, T→Left );
          if ( Height (T→Left) - Height ( T→Right ) == 2 )
              if (X < T→Left → Element )
                    T= SingleRotate WithLeft (T);
              else
                    T= DoubleRotate WithLeft (T);
        }
```

```
else if ( X > T -> Element )
{
    T -> Right = Insert ( X, T -> Right );
    if ( Height (T -> Right - Height (T -> Left ) == 2 )
        if ( X > T -> Right -> Element ).
            T = SingleRotatewith Right (T);
        else
            T = DoubleRotatewith Right (T);
}
T -> Height = Max ( Height (T -> Left), Height (T -> Right)
return T;
}
```

Deletion:

Deletion of a node in an AVL tree is similar to that of binary search trees. But after every deletion balance condition is checked to rebalance the tree..

Consider the AVL tree and delete 72 from it

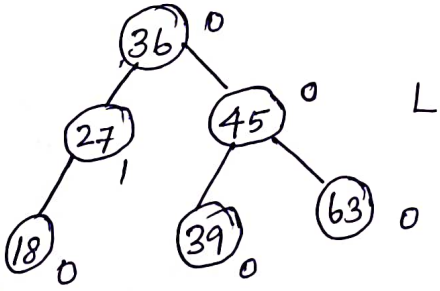


After deletion, node 45 is violated

There are 2 cases,

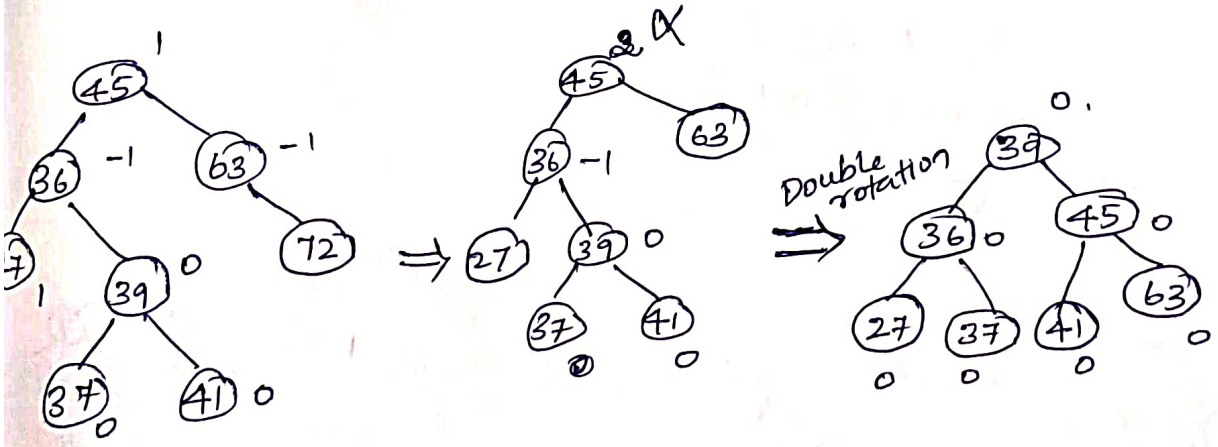
- Let B be the left or right of violated node α. If B is 0 or 1 then single rotation is applied.
- If B is -1, then double rotation is performed.

In the above example, B has 1 hence single rotat is performed.



LL rotation is applied on violated node (45)

Consider the below tree and delete 72



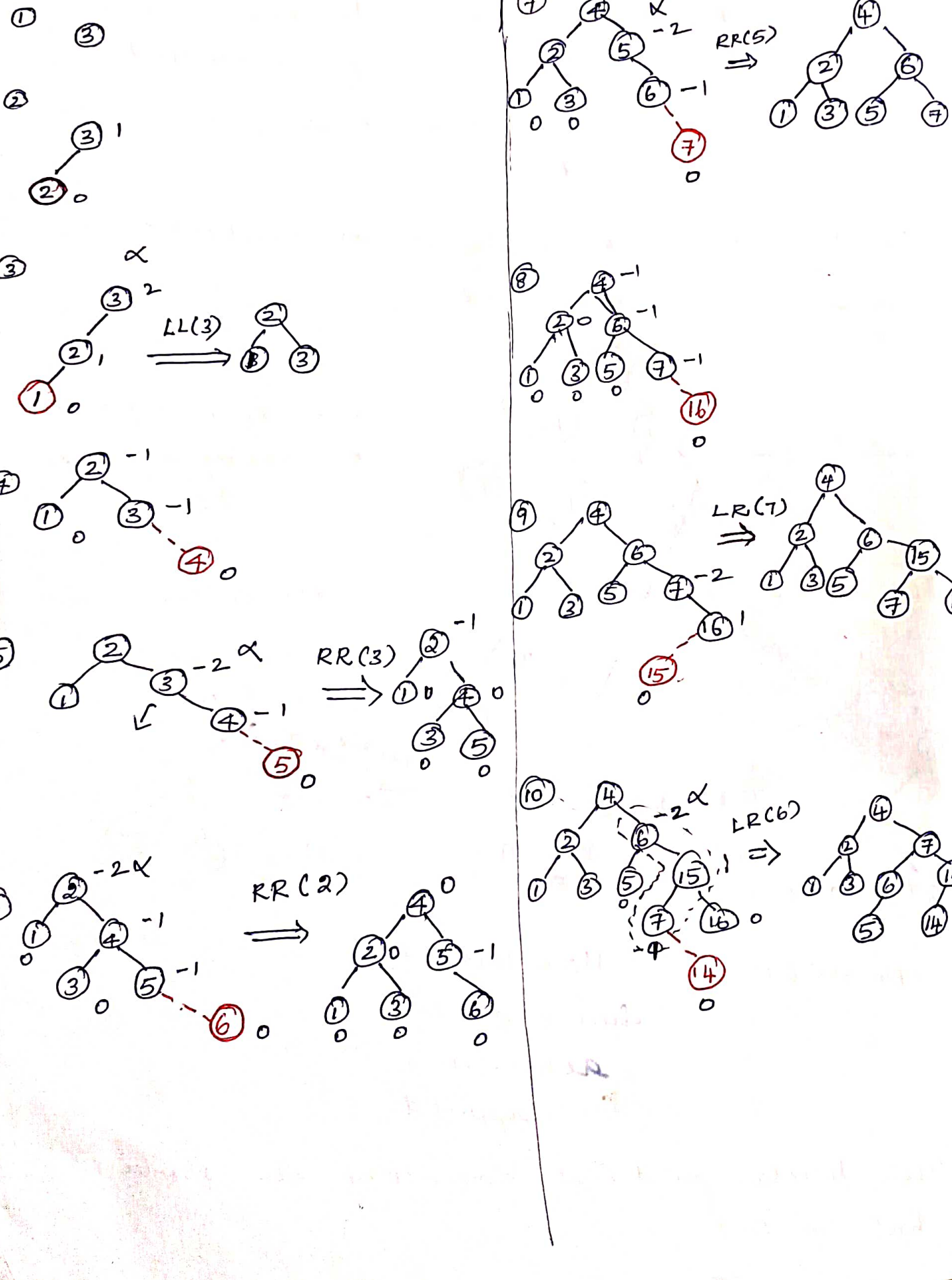
Delete (72)

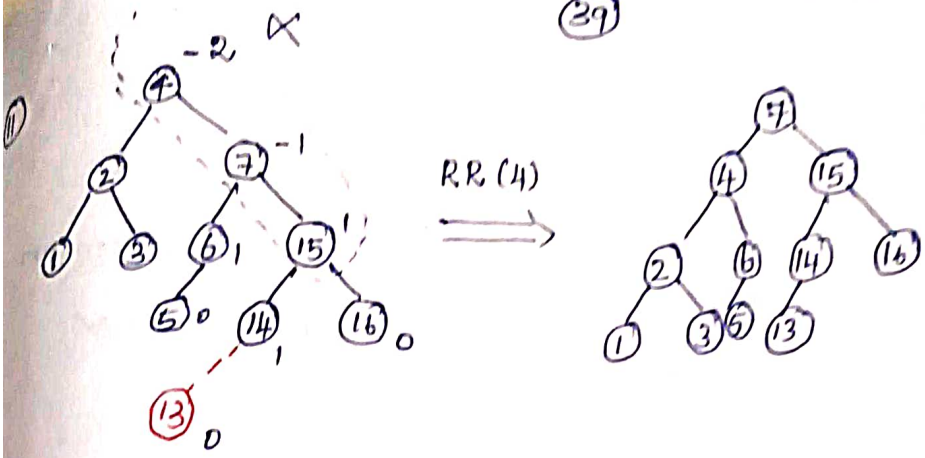
Here B is -1, hence double rotation is applied.

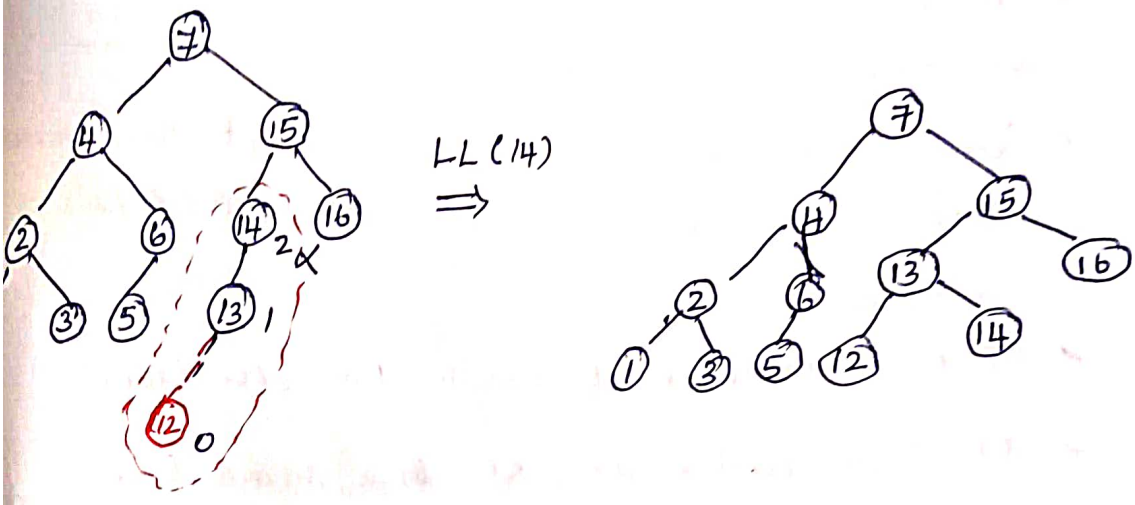
FindMin, FindMax and Find operations are similar to that of BST.

Construction of an AVL tree:

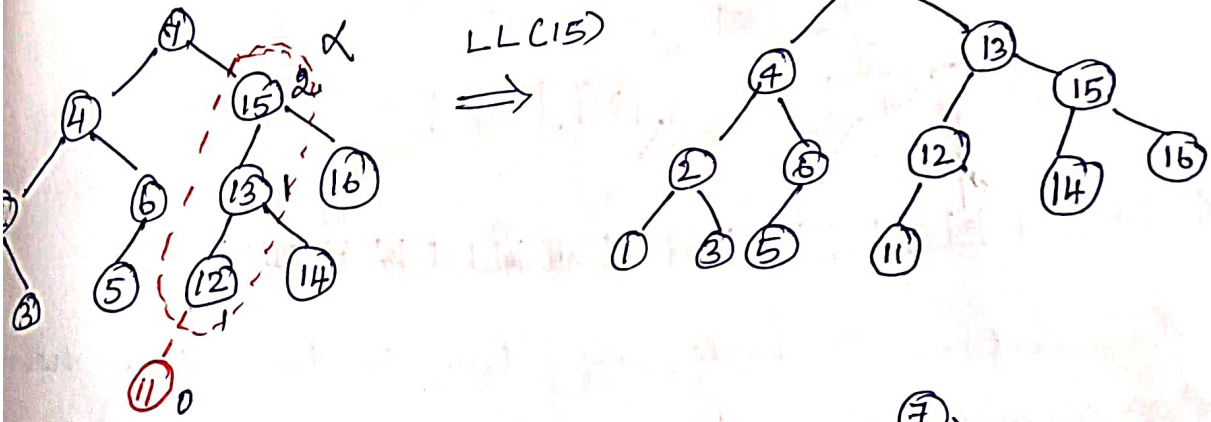
3, 2, 1, 4, 5, 6, 7, 16, 15, 14, 13, 12, 11, 10.

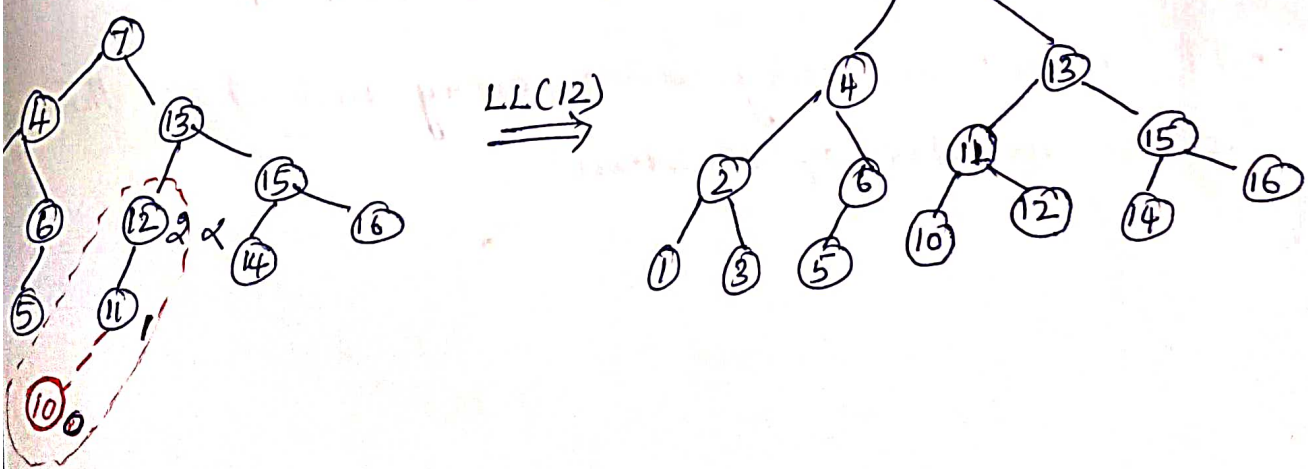






Insert 11





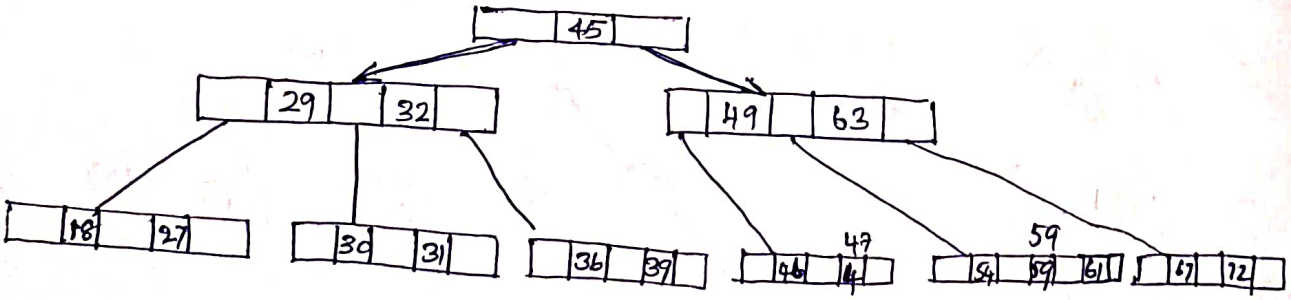
B-TREES

- B-tree is a specialized M-way search tree.
- A B-tree of order m can have a maximum of m-1 keys and m pointers to its sub tree

Properties:

* Every node in the B tree has at most 'm' children. (maximum)
* Every node in the B tree except the root node and leaf node has atleast m/2 children.
* Root node has at least two children
* All leaf nodes are at the same level.

Eg: B tree of order 4



- An internal node in the B tree can have n number of children, where $0 \leq n \leq m$.
- It is not necessary that every node has the same number of children.

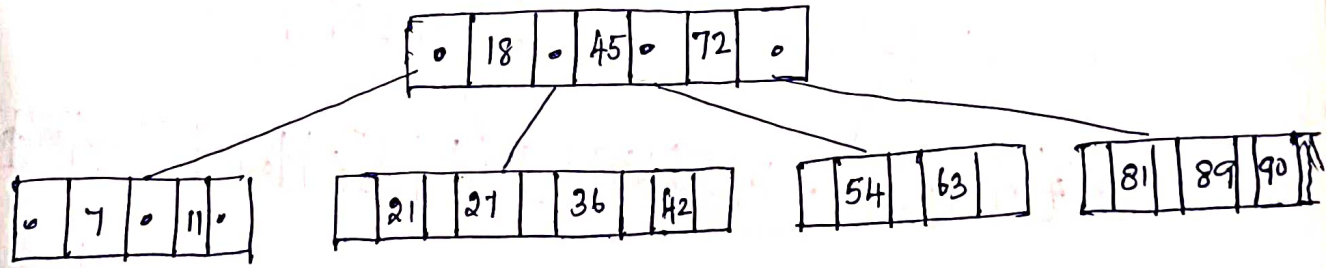
while performing insertion and deletion operations in a B-tree, the number of child nodes may change In order to maintain a minimum number of children, the internal nodes may be joined or plit.

Insertion:

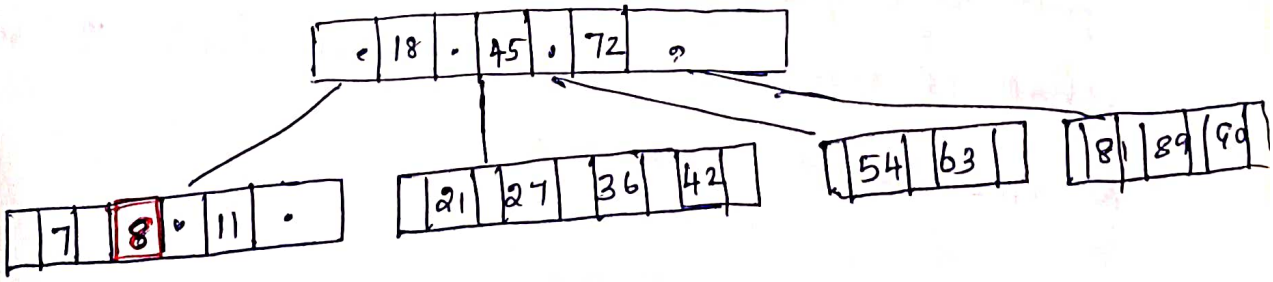
All the insertions are done at the leaf node

Cases:

case 1: If the leaf node is not full, ie it contains less than m−1 key values then insert the element in the node keeping the node's elements ordered.

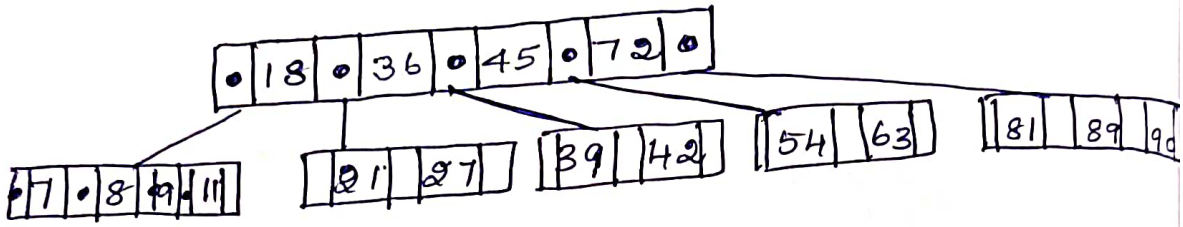


Insert 8



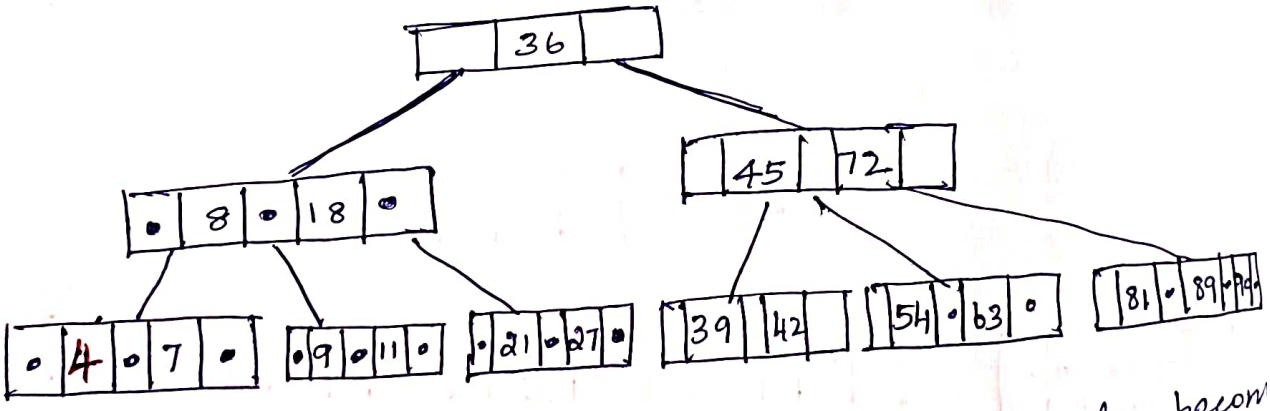
case 2: If the leaf node is full and root is not full, split the node into two nodes and move the median to its root.

Insert 39 in the above tree



case 3: If the leaf node full and root is also full.

Insert 4 in the above tree



When 4 is inserted, the leaf node becom full and pushed the median to root, since that is also full, it was split into 2 and a new node is created

Deletion:

Cases:

Case 1: If the leaf node contains more than the minimum number of key values, then delete the value

Case 2: If the leaf node does not contain $m/2$ elements, then fill the node by taking an element from the left or right sibling.

(a) If the left sibling has more than the minimum number of key values, push its largest key to its parent's node and pull down the intervening element from the parent node to the leaf node where the key is deleted.

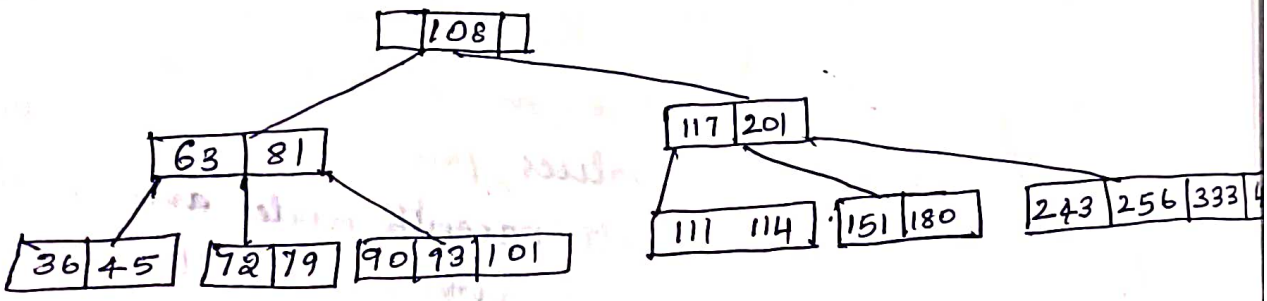
(b) If the right sibling has more than the minimum number of key values, push its smallest key into its parent node and pull down the intervening element from the parent node to the leaf node where the key is deleted.

Case 3: If both left and right siblings contain minimum, then create a new leaf node by combining the two leaf nodes and intervening element of parent node.

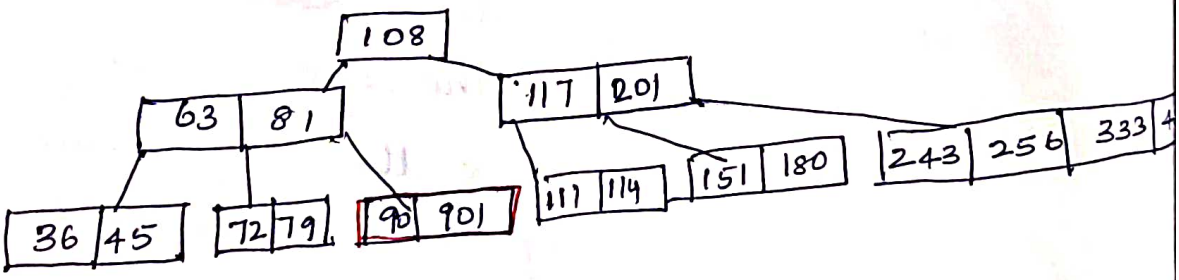
If pulling the intervening element from the parent node leaves it with less than minimum number of keys in the node, then propagate the process upwards.

Case 4: To delete an internal node, promote the successor or predecessor of the key to be deleted.

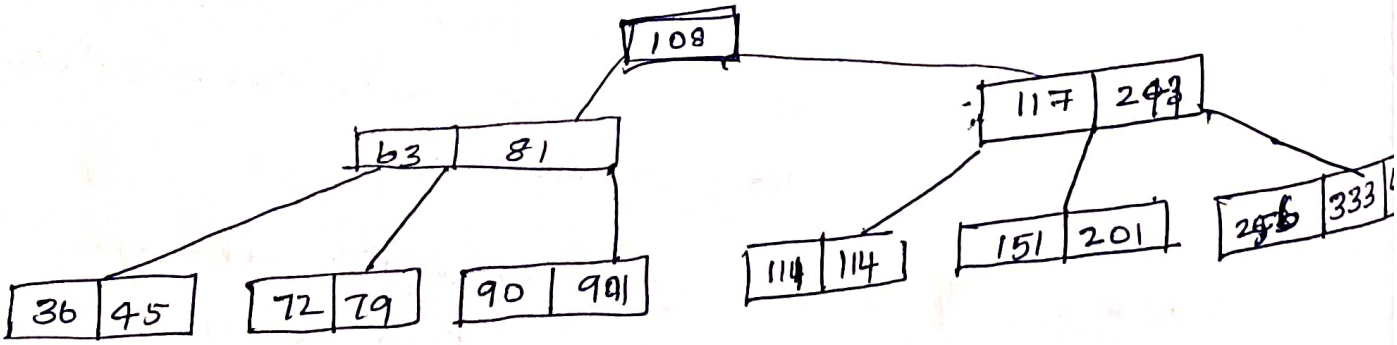
~~Case 1:~~ Example for case 1:



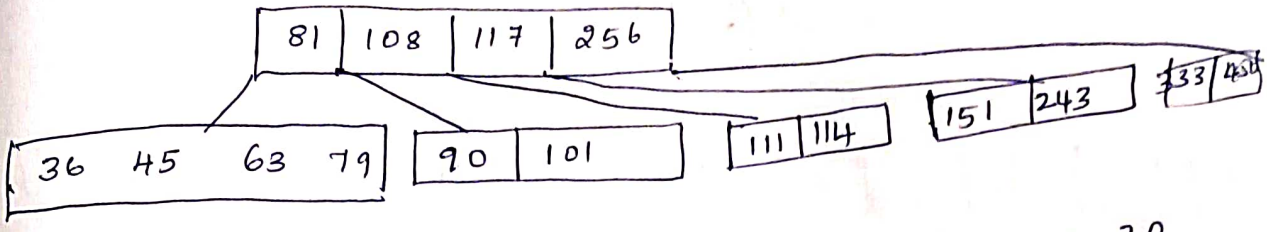
Delete 93



Example for case 2: Delete 180

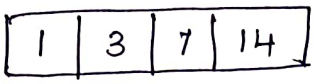


Example for case 3: Delete 72

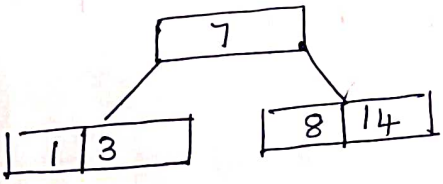


Construct a B-tree: 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20
26, 4, 16, 18, 24, 25 & 19

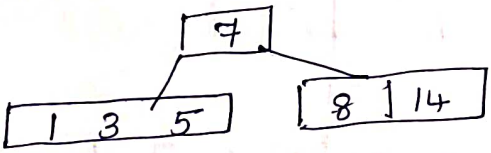
Order of 5



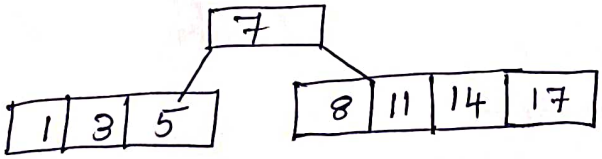
Insert 8



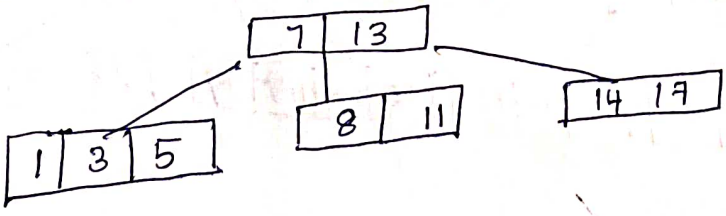
Insert 5



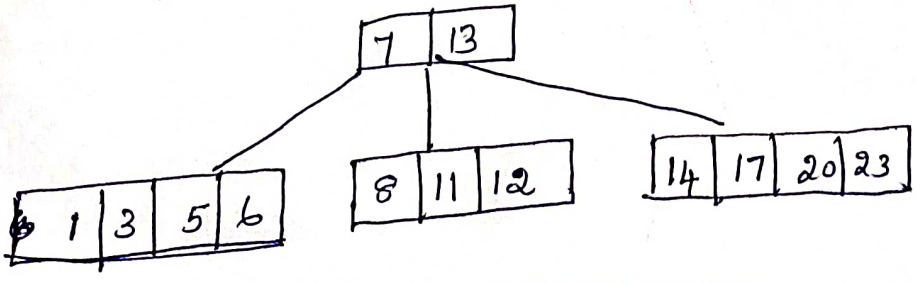
Insert 11, 17



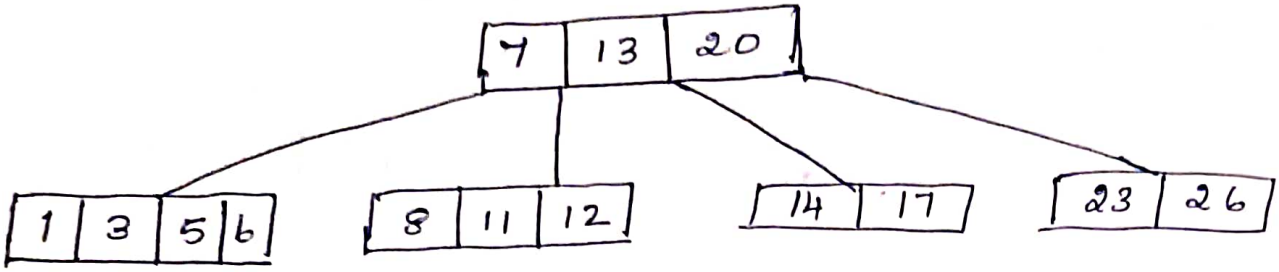
Insert 13, split right into 2



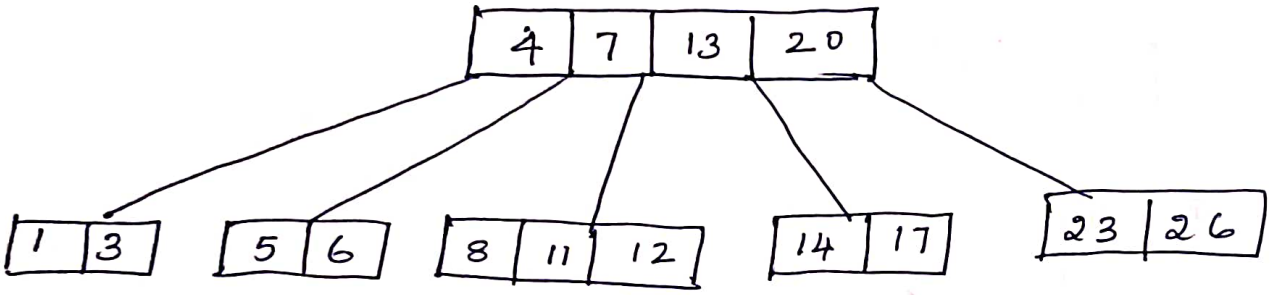
Insert 6, 23, 12, 20



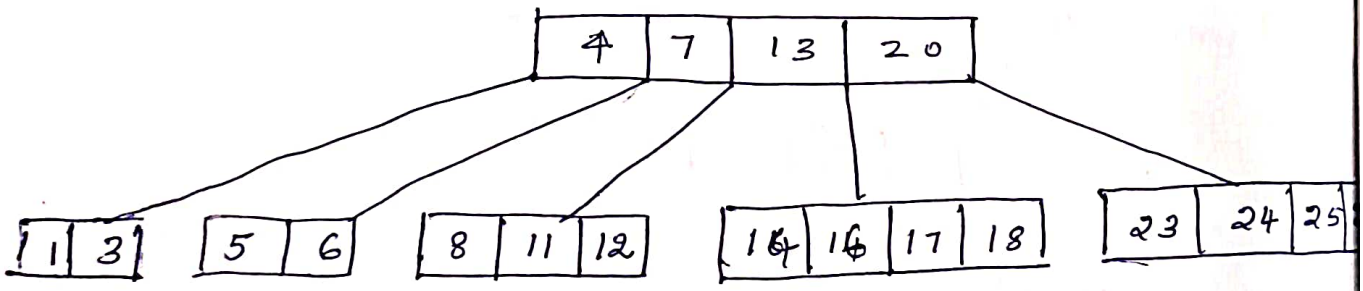
7. Insert 26



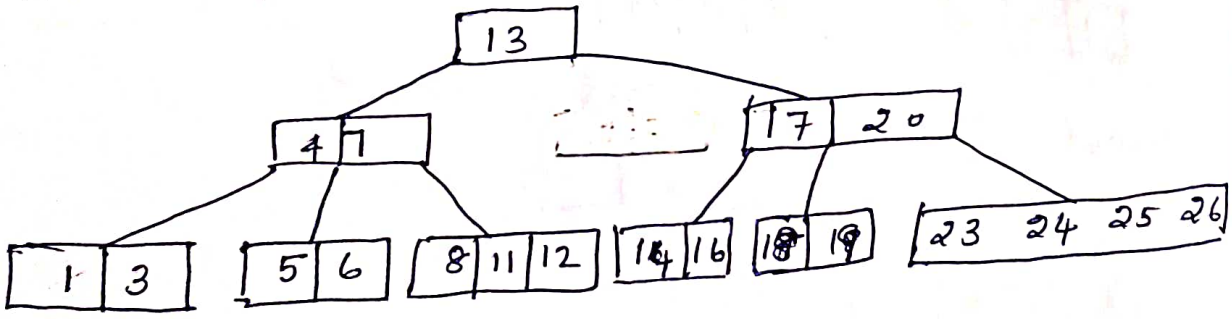
8. Insert 4



9. Insert 16, 18, 24, 25



10. Insert 19

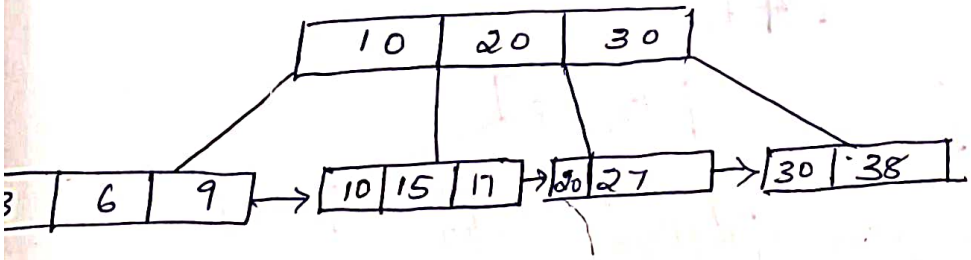


- It can be used to perform a wide range of queries easily as leaves are linked to nodes at the upper level.
- Height of a tree is less and balanced
- Supports both random and sequential access to records.
- Keys are used for indexing.

Insertion:

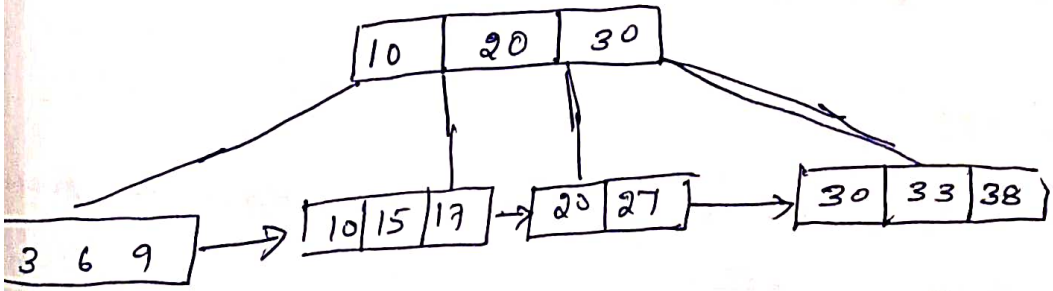
Algorithm:

1. Insert a new node in leaf
2. If the leaf node overflows, split the node and copy the middle element to next index node.
3. If the index node overflows, split that node and move the middle element to next index page.



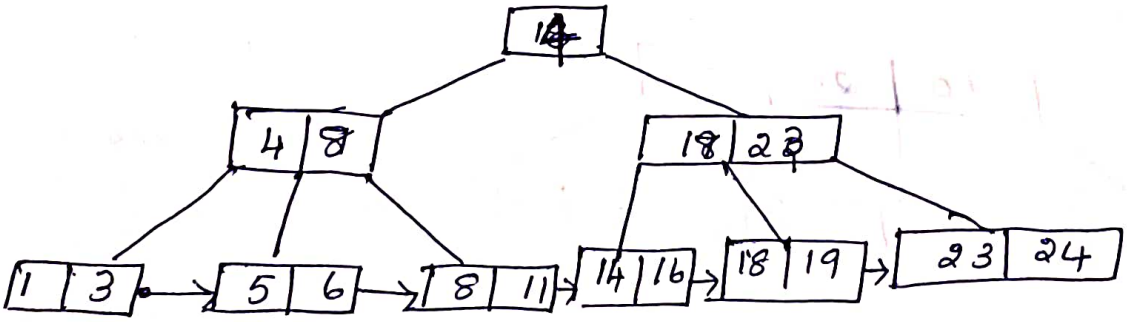
B+ tree of order 4

Insert (33)



B+ TREES:

- A B+ tree is a variant of B tree which stores sorted data.
- B+ tree can store both keys and records in its interior nodes. But B+ tree stores all the records at the leaf level of the tree only keys are stored in the interior nodes.
- Leaves are linked to one another in a linked list.
- B+ trees are used to store large amounts of data that cannot be stored in the main memo Leaf nodes are stored on the secondary storage and internal nodes are stored in the main memory.
- B+ trees stores data only in the leaf nodes and all other nodes store index values.



Eg: B+ tree of order 3.

Advantages:

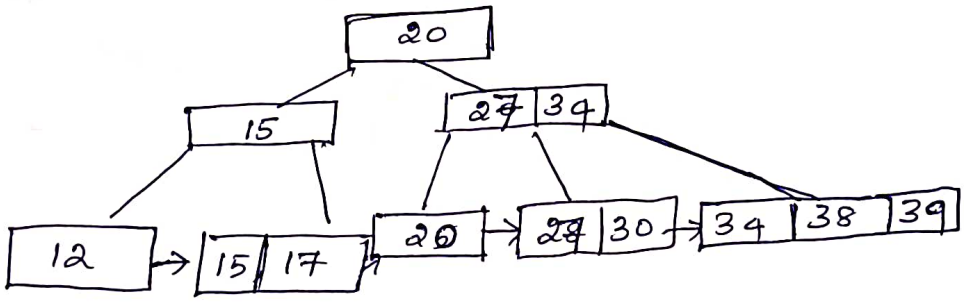
- Records can be fetched in equal num of disk accesses

Deletion:

- Deletion is always done from a leaf node
- If deleting a data element leaves that node empty, then the neighbaing nodes are are examined and merged.

---

1. Delete the key and data from leaves
2. If the leaf node underflows, merge that node with the sibling and delete the key in bbtwen them.
3. If the index node underflows, merge that node with sibling and move down the key in between them.



B+ tree

Delete (39)