Marwadi University
Marwadi Chandarana Group

UNIT 3

Non-Linear Data Structure-Trees

Marwadi University
Marwadi Chandarana Group

CONTENTS

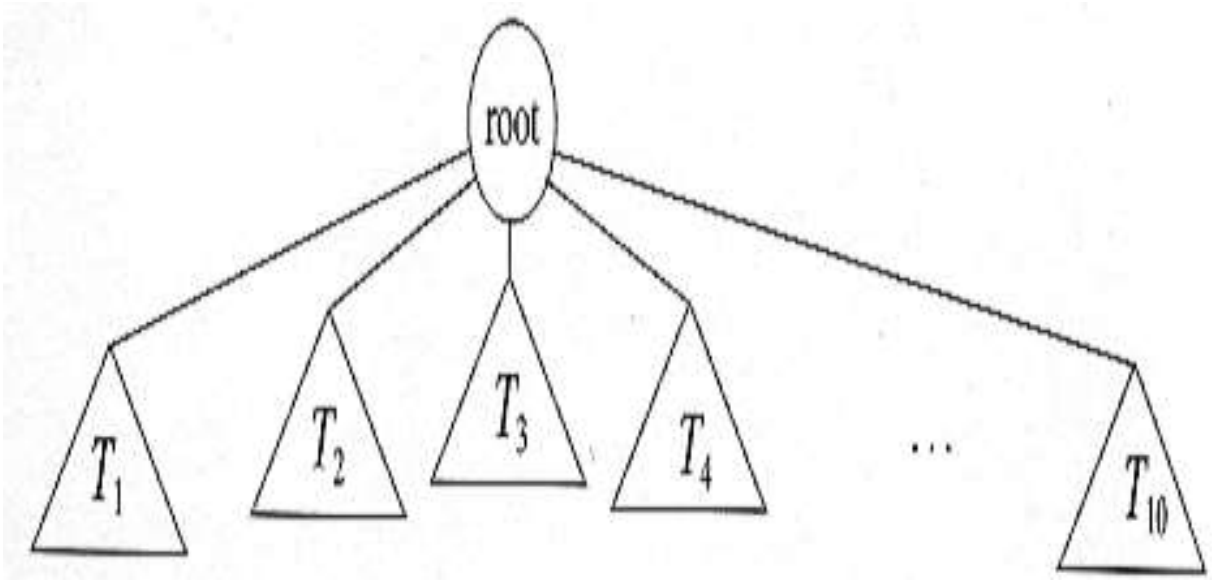
- Tree definitions and their concepts,
- Representation of binary tree,
- Binary tree traversal methods (Inorder, postorder, preorder),
- Binary search trees,
- Reconstruction of a binary tree from traversal
- Method to Convert a general tree to binary tree,
- Threaded binary tree,
- Applications of Trees,
- Balanced tree and its mechanism,
- AVL tree,
- Weight Balanced Trees,
- B Tree and B+ Tree

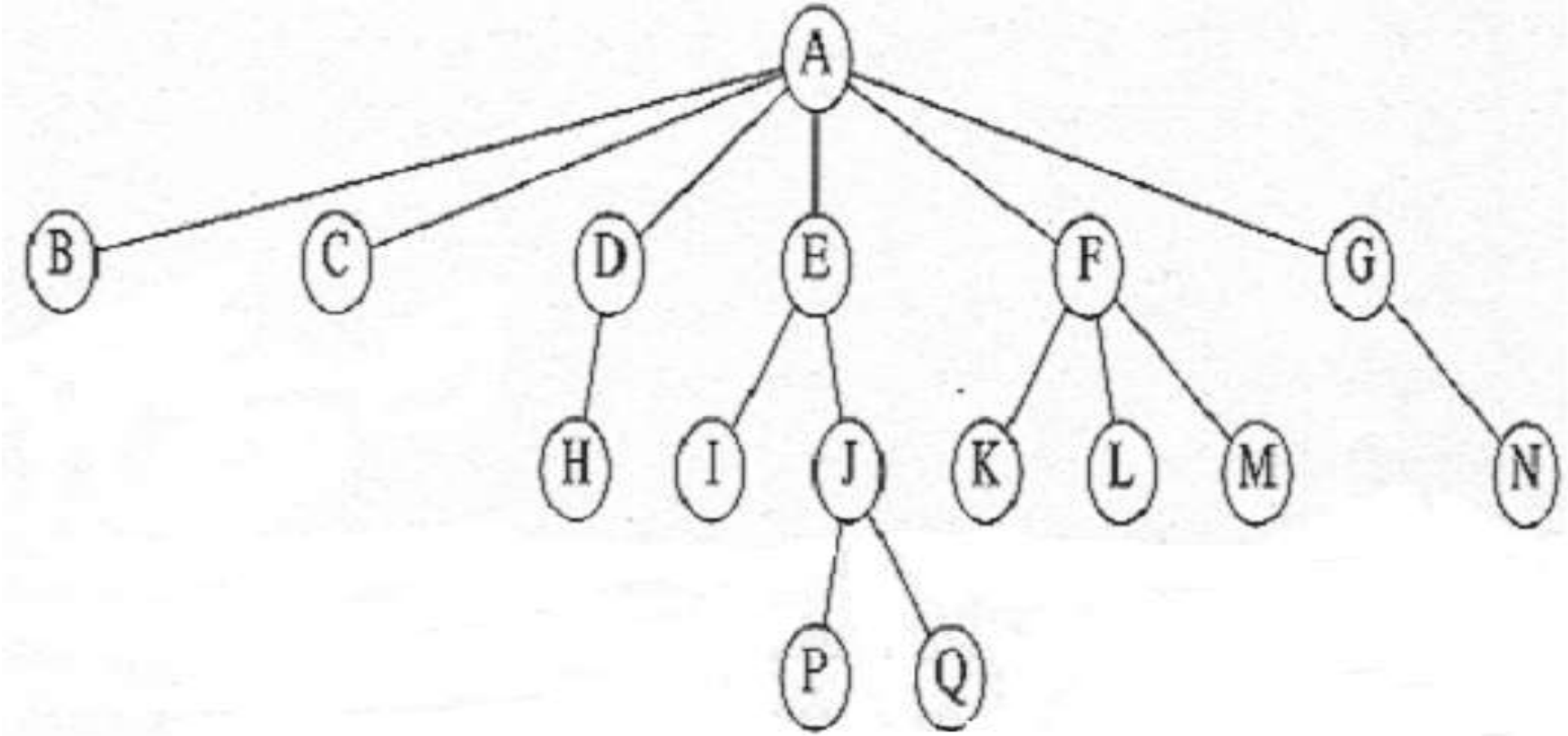
Non-Linear Data Structure

- Not stored in sequential order
- Branches to more than one node
- Can't be traversed in a single run
- Data members are not processed one after another

TREES/ GENERAL TREES

- Non linear data structure **which organizes data in hierarchical structure**
- A tree is either
  - empty (no nodes), or
  - If not empty, a tree consists of a distinguished root node (r), and zero or more nonempty subtrees T_1, T_2, ..., T_k





Terminology

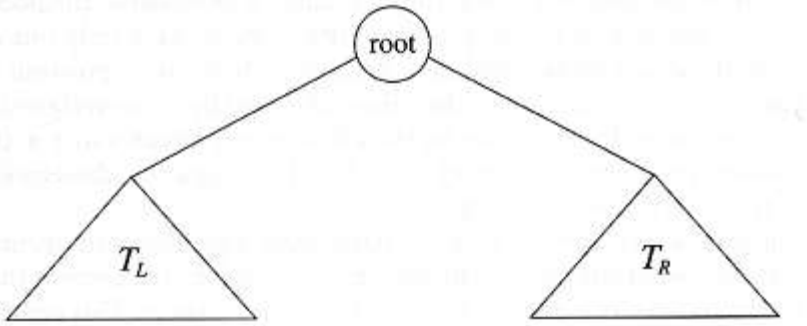
- Root
  - Base node (vertex) of the tree
- Leaves
  - Nodes with no children / Nodes (i.e. with outdegree zero)
- Internal nodes
  - Nodes that are not root or leaf
- Indegree
  - Number of edges (branches) arriving at that vertex.
- Outdegree
  - Number of edges leaving that vertex

- Degree
  - Number of branches associated with a node
- Parent
  - Node with out degree greater than zero. Every node except the root has one parent .
- Child
  - A node which has a predecessor
- Sibling
  - Nodes with same parent
- Path
  - Sequence of nodes in which each node is adjacent to next

- Ancestor and descendant
  - If a path exists from node p to node q, where node p is closer to the root node than q, then p is an ancestor of q and q is a descendant of p.
- Level of a node
  - Distance of node from root. Root at level 0.
- Depth of a tree
  - Maximum level of any leaf in a tree.
- Height of tree
  - The height of a tree is the level of the leaf in the longest path from the root plus 1. Height of empty tree is -1.
  - It is a maximum number of nodes in longest path of tree.

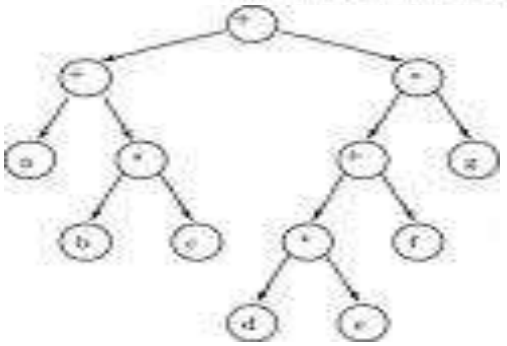
Binary Trees

- A tree in which no node can have more than two children

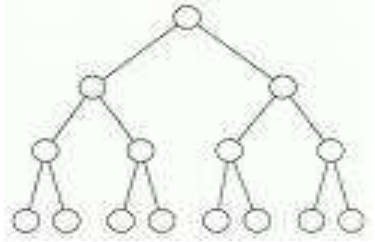


- M-ary Tree
  - A tree in which no node can have more than M children

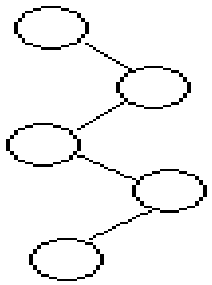
- Strict (Full) Binary tree
  - Every node has zero or two children. n leaves => 2n-1 nodes.



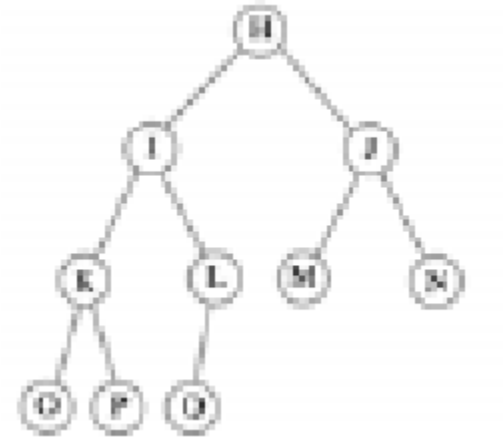
- Complete (perfect) binary tree
  - Strict binary tree in which all leaves are at the same depth d. Total no. of nodes 2^{d+1}-1. For every level d, 2^{d} nodes.



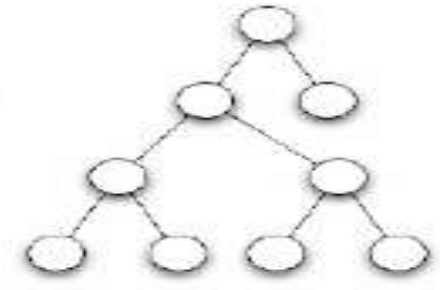
- Degenerate tree
  - Every node has only one child. It behaves like a linked list data structure.

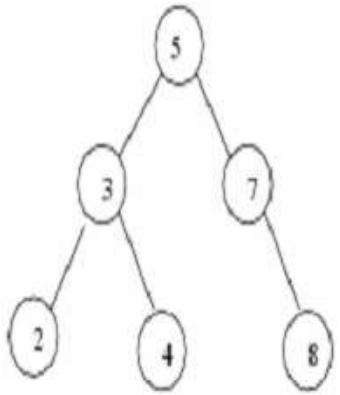


- Almost complete binary tree
  - The tree is almost complete if all its levels, except possibly the last, has max. no. of nodes, and if all nodes at the last level appear as far left as possible.
  - Each node that has a right child also has a left child.
  - A node having a left child does not require a node to have a right child.
  - Leaves are at level d or d-1



Almost complete binary tree





Not Almost complete binary tree

Marwadi University
Marwadi Chandarana Group

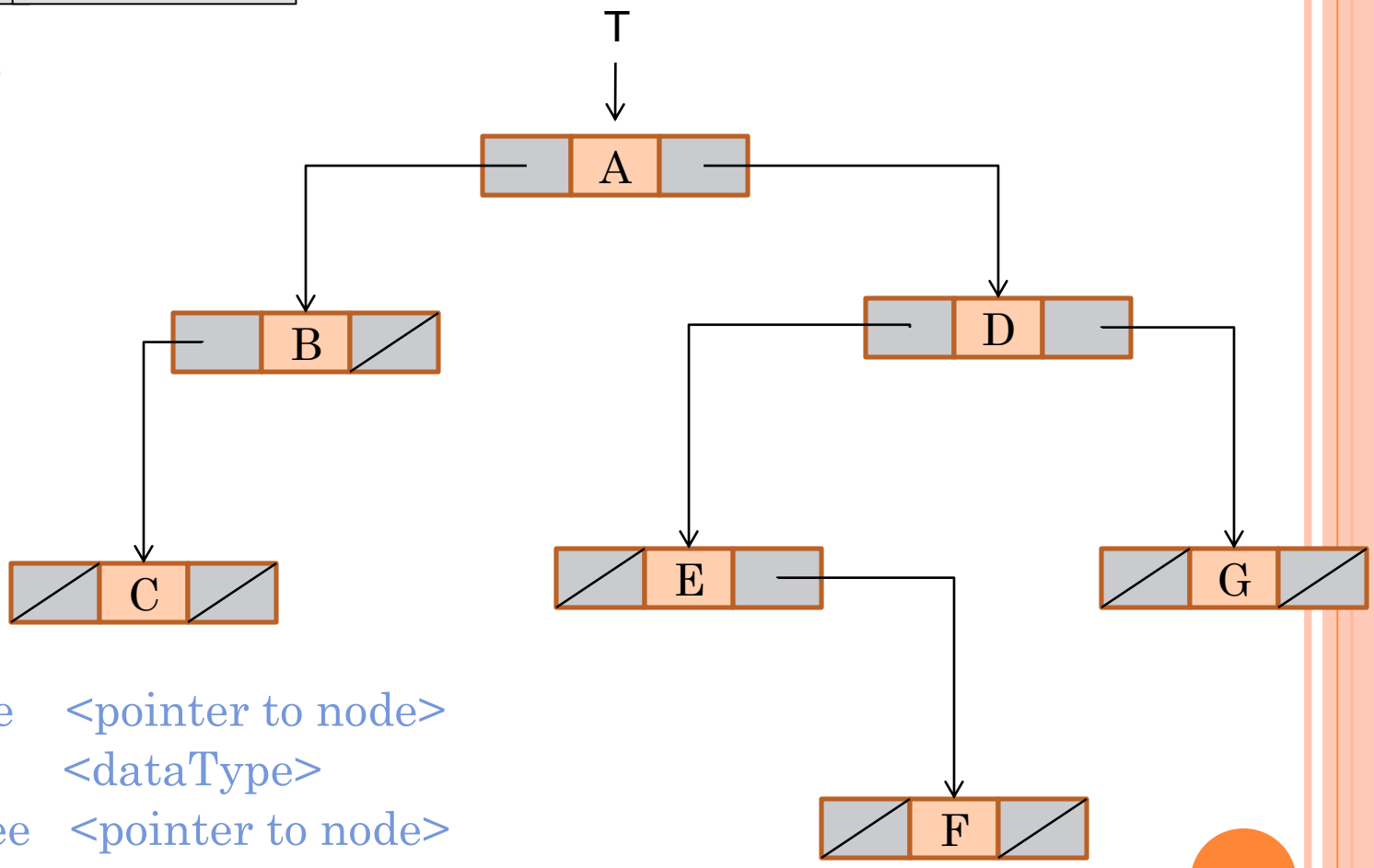
Storage Representation of Binary Tree

- Linked Storage Representation
- Array Representation

Linked Representation

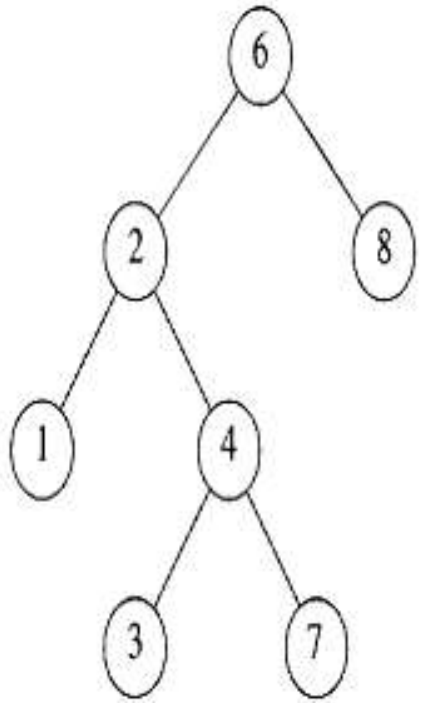
Pointer to left subtree	Data	Pointer to right subtree

Node




```
Node
   leftSubtree    <pointer to node>
   data           <dataType>
   righSubtree    <pointer to node>
End node
```

Array Representation



To represent a binary tree of depth **'n'** using array representation, we need one dimensional array with a maximum size of $2^{n+1} - 1$.

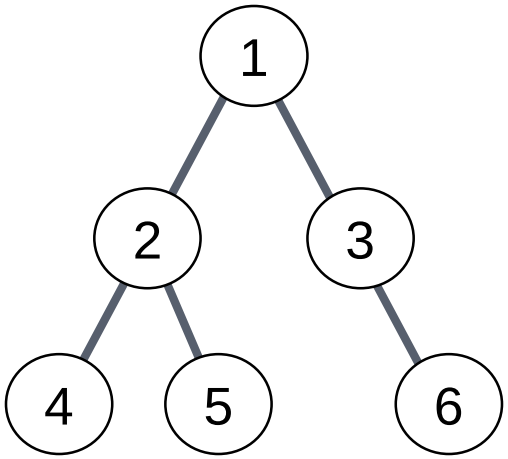
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
6	2	8	1	4	-	-	-	-	3	7	-	-	-	-

Binary Tree Traversal

- A binary tree traversal requires that each node of the tree be processed once and only once in a predetermined sequence
- Breadth First Traversal
  - Traverse the tree horizontally from level 0 to last level in tree.
- Depth First Traversal
  - Process all descendants of a child before going on to the next child
    - Preorder Traversal
    - Inorder Traversal
    - Postorder Traversal

Preorder

- Visit the node.
- Traverse the left subtree in preorder
- Traverse the right subtree in preorder



Root →Left→ Right

preorder : 1 2 4 5 3 6

Algorithm preOrder (root <tree ptr>)

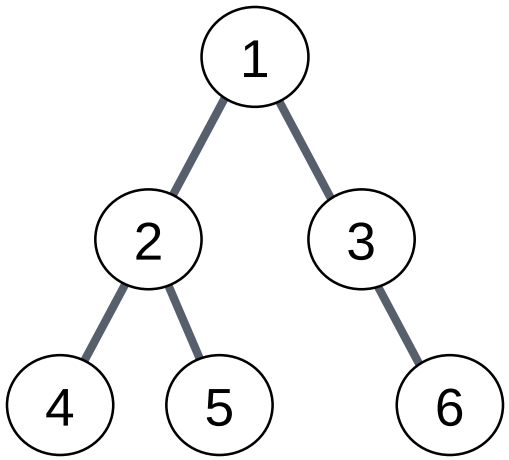
Pre: root points to root of the tree

Return : Traverses the tree in preorder

1. if (root <>NULL)
    1. process(root->data)
    2. preOrder(root->left)
    3. preOrder(root->right)
2. return

Inorder

- Traverse the left subtree in inorder.
- Visit the node.
- Traverse the right subtree in inorder.



Left→ Root → Right

Inorder : 4 2 5 1 3 6

Algorithm inOrder (root <tree ptr>)

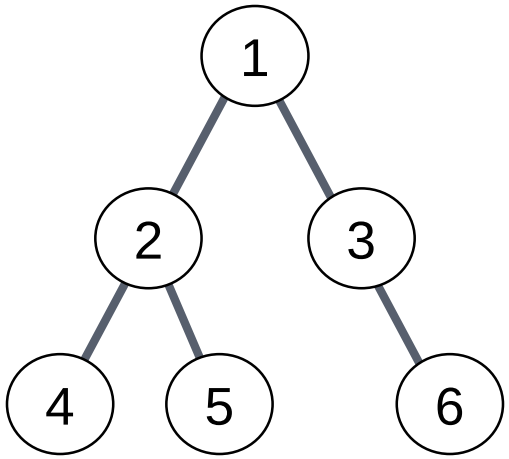
Pre: root points to root of the tree

Return : Traverses the tree in inorder

1. if (root <>NULL)
    1. inOrder(root->left)
    2. process(root->data)
    3. inOrder(root->right)
2. return

POSTORDER

- Traverse the left subtree in postorder.
- Traverse the right subtree in postorder.
- Visit the node.



Left→ Right → Root

Postorder : 4 5 2 6 3 1

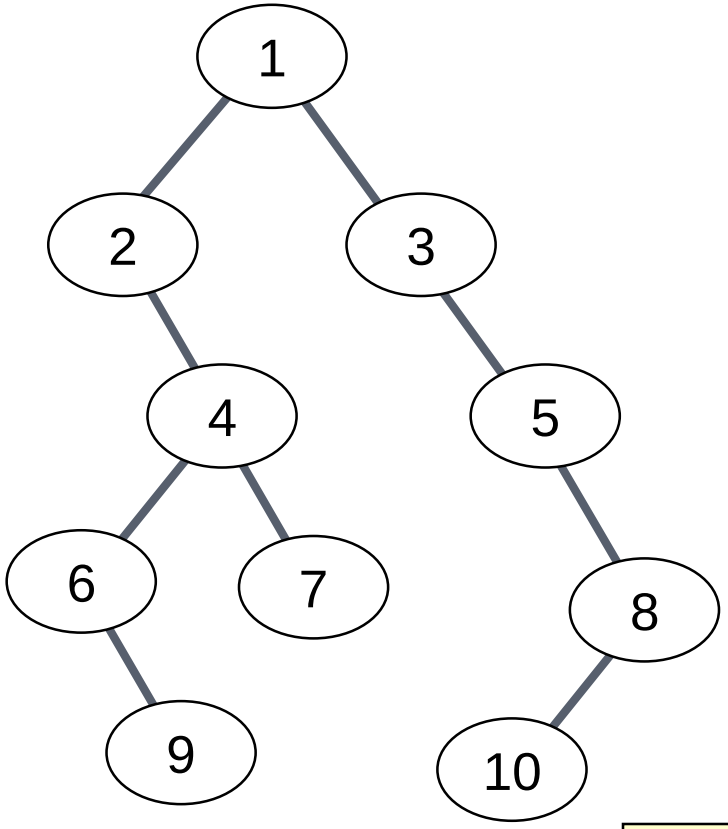
Algorithm postOrder (root <tree ptr>)

Pre: root points to root of the tree

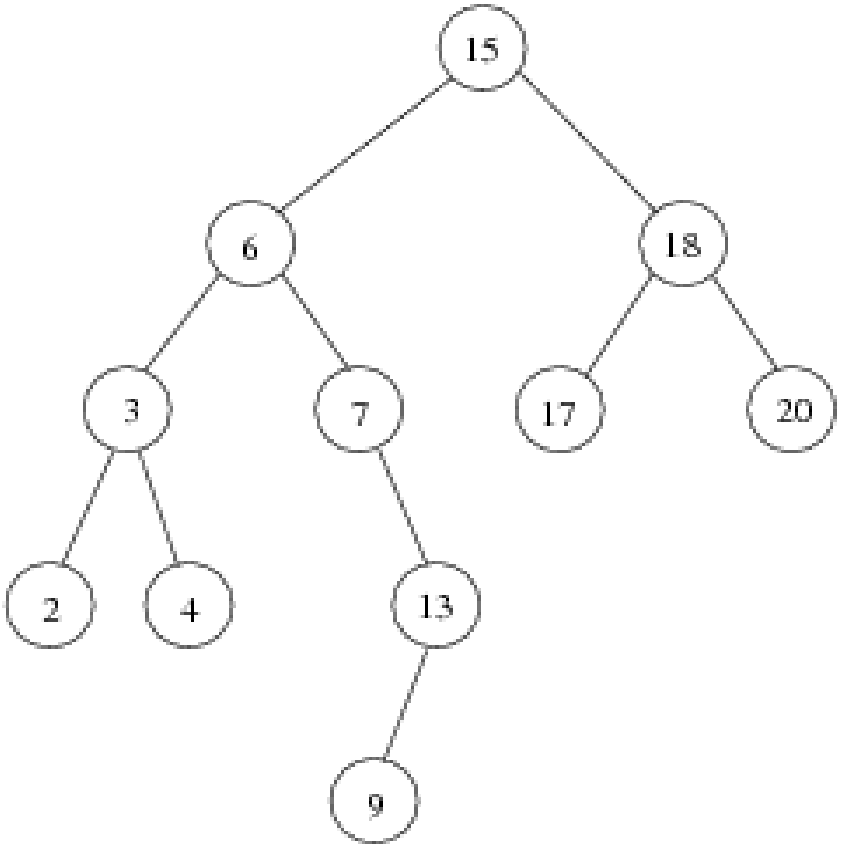
Return : Traverses the tree in postorder

1. if (root <>NULL)
    1. postOrder(root->left)
    2. postOrder(root->right)
    3. process(root->data)
2. return

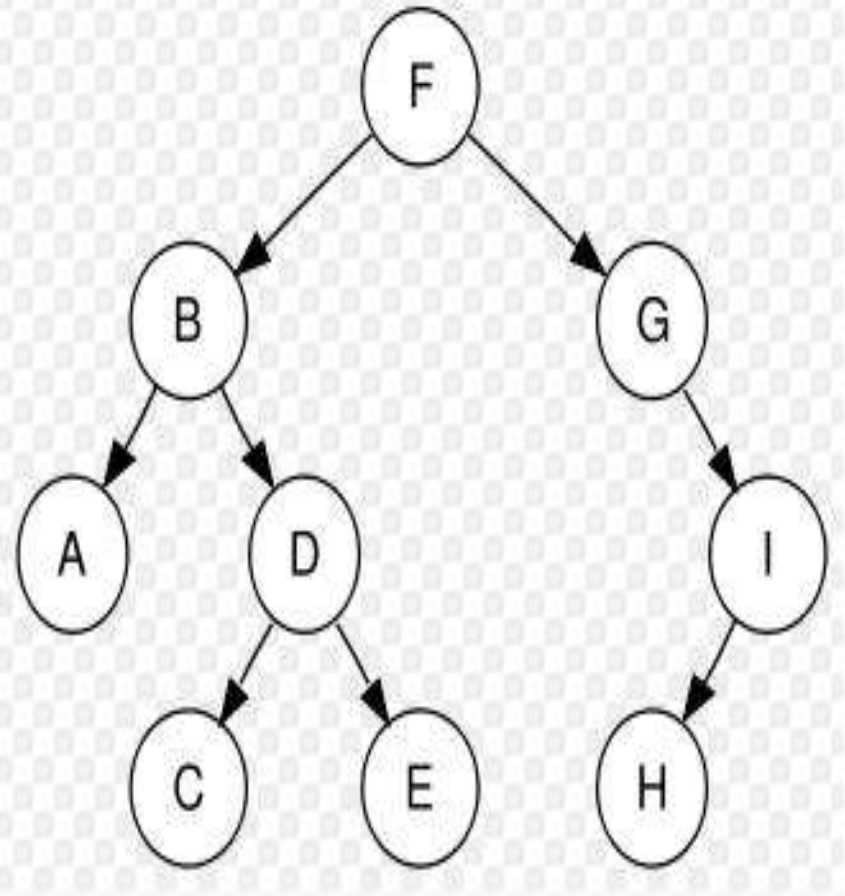
Traverse the following binary tree in preorder, inorder and in postorder.



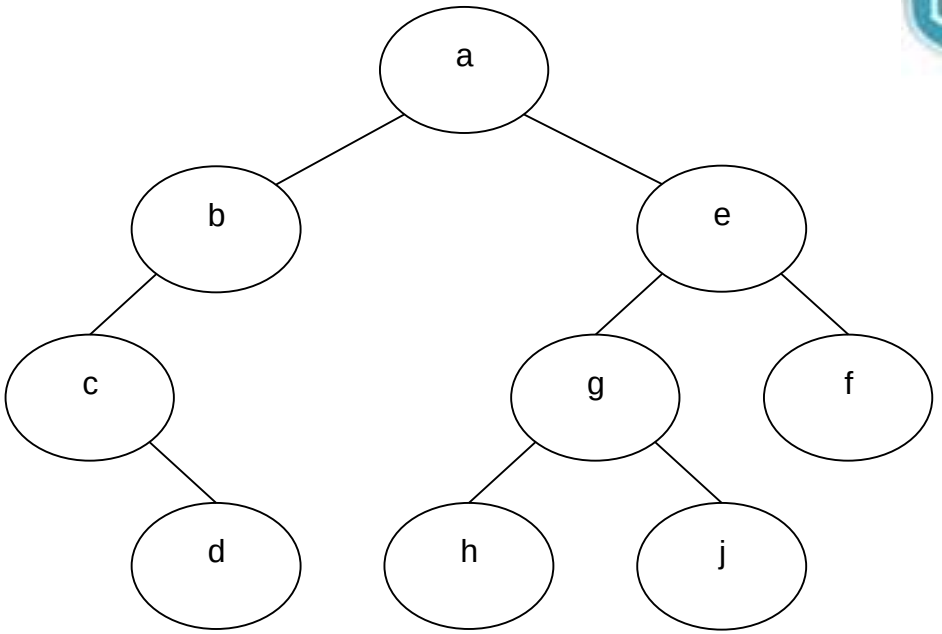
preorder : 1, 2, 4, 6, 9, 7, 3, 5, 8, 10
inorder : 2, 6, 9, 4, 7, 1, 3, 5, 10, 8
postorder : 9, 6, 7, 4, 2, 10, 8, 5, 3, 1



preorder : 15, 6, 3, 2, 4, 7, 13, 9, 18, 17, 20
inorder : 2, 3, 4, 6, 7, 9, 13, 15, 17, 18, 20
postorder : 2, 4, 3, 9, 13, 7, 6, 17, 20, 18, 15



preorder : F, B, A, D, C, E, G, I, H
inorder : A, B, C, D, E, F, G, H, I
postorder : A, C, E, D, B, H, I, G, F



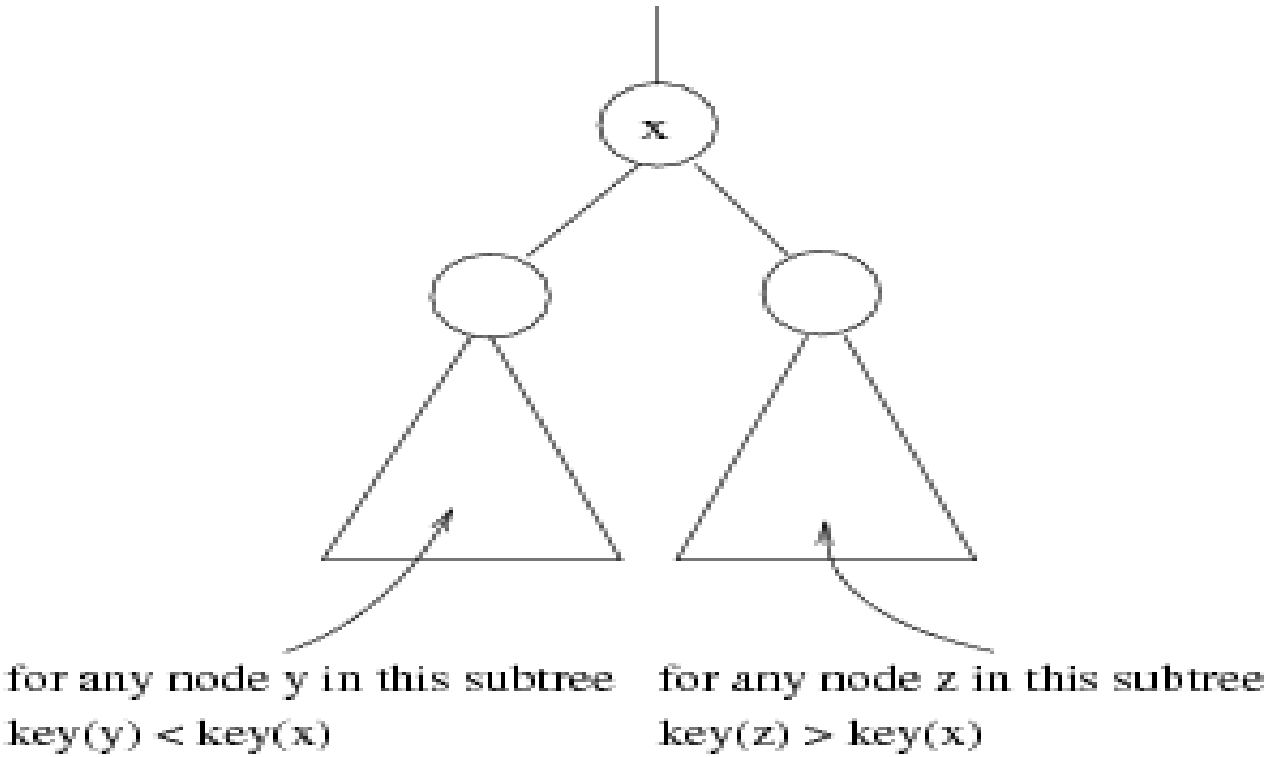
preorder :a,b,c,d,e,g,h,j,f.
inorder : c,d,b,a,h,g,j,e,f.
postorder :d,c,b,h,j,g,f,e,a.

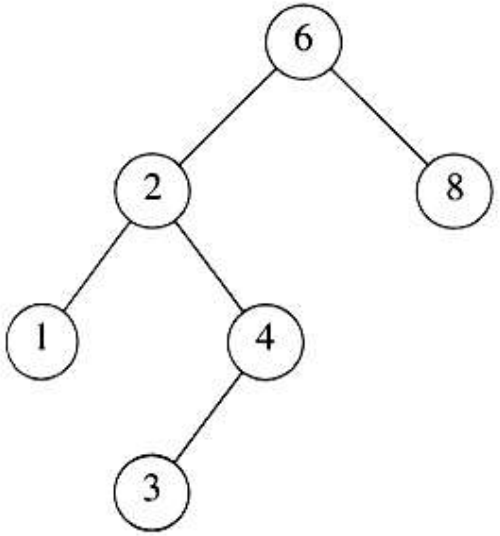
Marwadi University
Marwadi Chandarana Group

Binary Search Trees(BST)

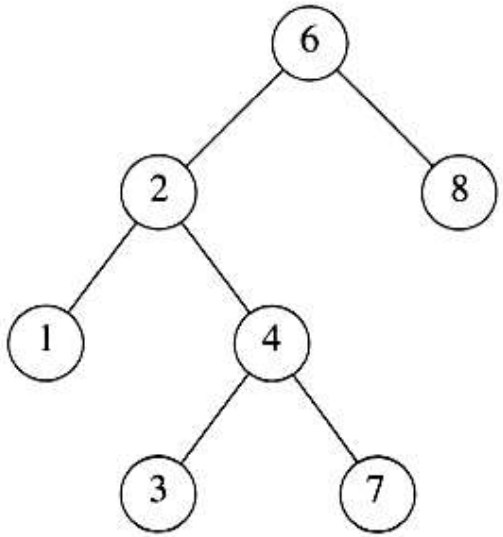
Binary Search Trees

- For every node X, all the keys in its left subtree are smaller than the key value in X, and all the keys in its right subtree are larger than the key value in X

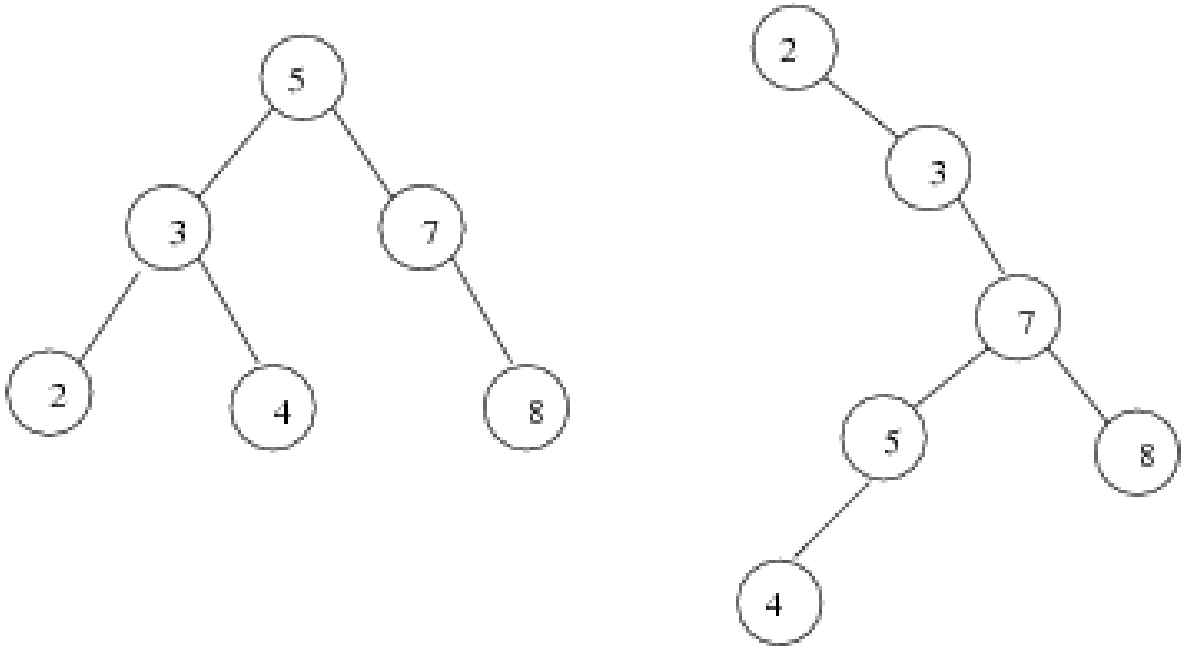




A binary search tree



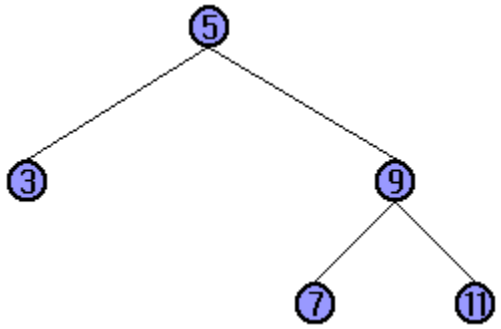
Not a binary search tree



Two binary search trees representing the same set:

Why should we use binary search trees ?

- Binary search tree can be considered as a self-sorting data structure.
- One does not have to search the entire tree for a particular item in the manner of linked list traversals.



OPERATIONS ON BST

- BST Traversals
  - Preorder
  - Inorder
  - Postorder
- Search
  - Find smallest node
  - Find largest node
  - Find specific node
- Insertion
  - Insertion take place at a leaf or a leaflike node, a node that has only one null branch

- Deletion
  - The node to be deleted has no children's
  - The node to be deleted has only right subtree
  - The node to be deleted has only left subtree
  - The node to be deleted has two subtree

Searching BST

- **Find smallest node**
  - The node with smallest value is the leftmost leaf node in the tree.
  - To find smallest node follow the left branches until we get to a leaf .
  - ALGORITHM:
    - findSmallestBST(val root<pointer>)
    - This algorithm find smallest node in a BST.
    - **Pre :** root is a pointer to a nonempty BST or subtree
    - **Return :** address of smallest node

```
1. if(root - >left == NULL)
      Return (root)
2. end if
3 return findSmallestBST(root - >left)
End findSmallestBST
```

- **Find largest node**
  - The node with largest value is the rightmost leaf node in the tree.
  - To find largest node follow the right branches until we get to a leaf .
  - ALGORITHM:
    - findLargest BST(val root<pointer>)
    - This algorithm find largest node in a BST.
    - **Pre :** root is a pointer to a nonempty BST or subtree
    - **Return :** address of largest node

      1. if(root - >right == NULL)

         Return (root)

      2. end if

      3 return findLargest BST(root - >right)

      End findLargest BST

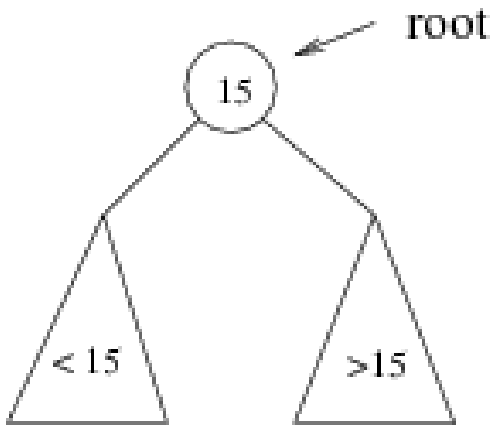
Minimum and maximum node in BST

```
TREE-MINIMUM (x)
while x->left <> NULL do
    x = x->left
return x
```

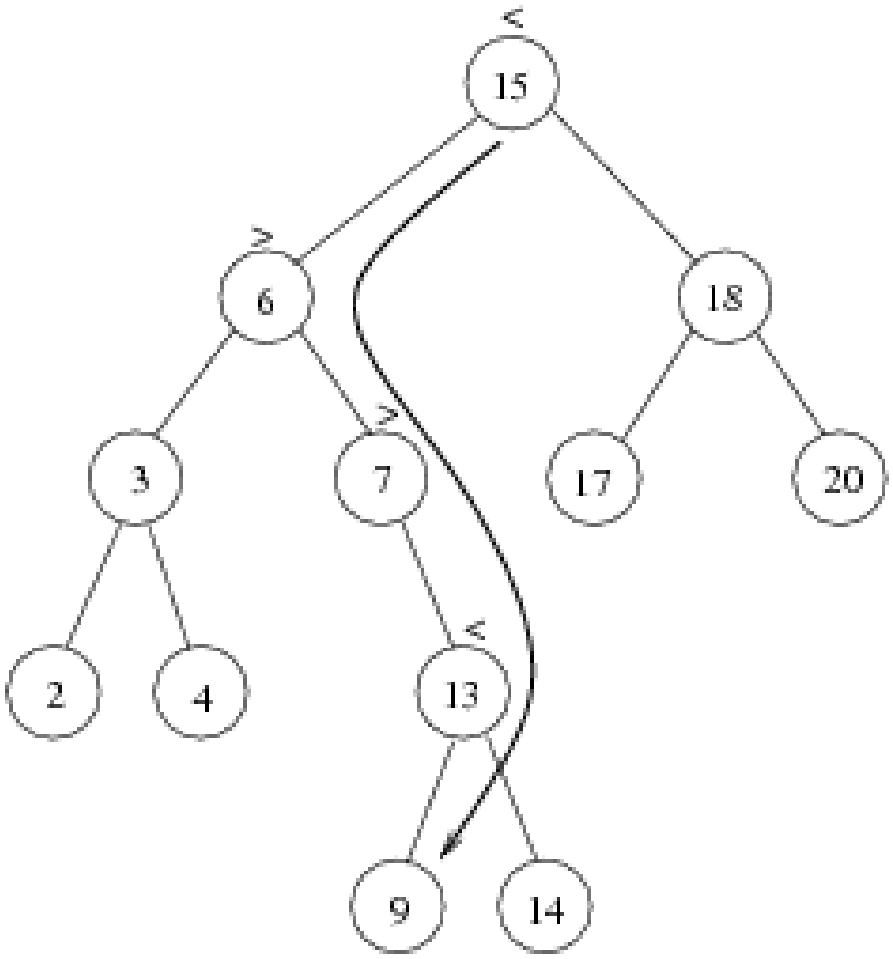
```
TREE-MAXIMUM (x)
while x->right<> NULL do
    x = x->right
return x
```

Searching BST

- If we are searching for 15, then we are done.
- If we are searching for a key < 15, then we should search in the left subtree.
- If we are searching for a key > 15, then we should search in the right subtree.
- Recursively repeat the same process, till you get your key or reach leaf level without finding key.



Example: Search for 9 ...



Algorithm recSearch (root <tree ptr>, target<keyType>)

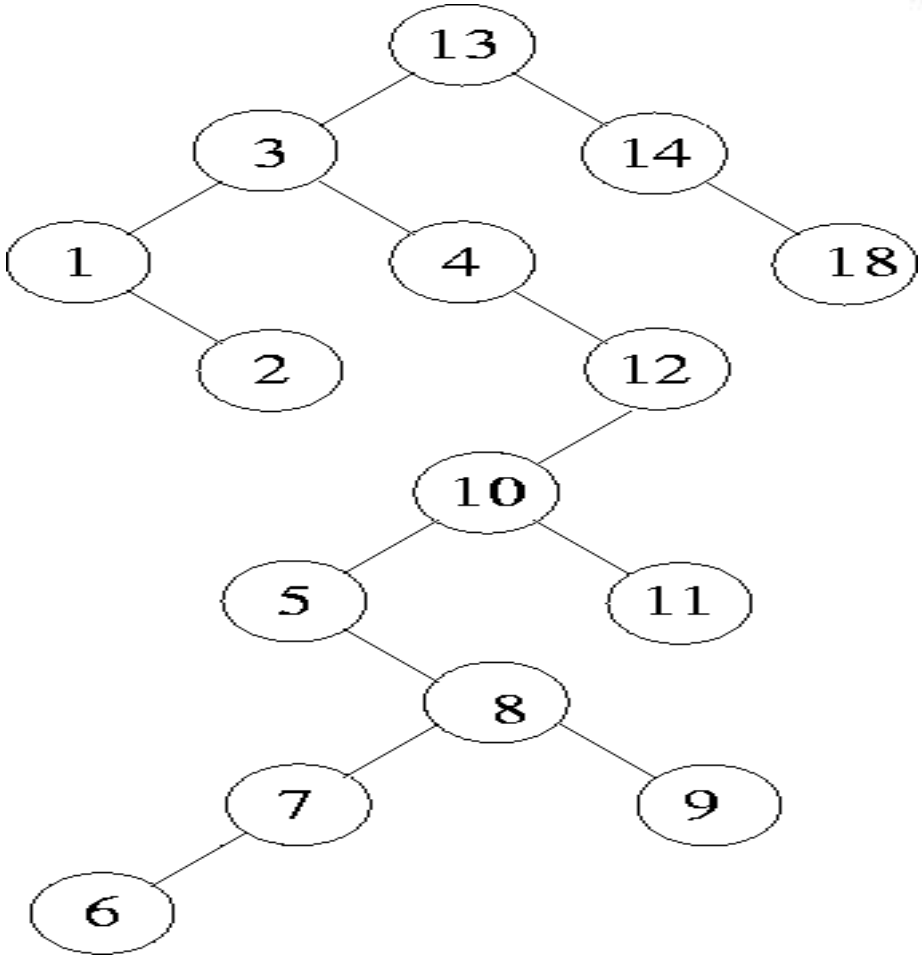
Pre: root points to root of the tree

Return : Node address where key is found, NULL if not found

1. if (root == NULL)
    1. return NULL
2. if (target < root->key)
    1. return recSearch (root->left, target)
3. else if (target > root->key)
    1. return recSearch (root->right, target)
4. else
    1. return root

Marwadi University
Marwadi Chandarana Group

Question: Construct BST .
13, 3, 4, 12, 14, 10, 5, 1, 8, 2, 7, 9, 11, 6, 18



Write Inorder, Preorder or Postorder Traversal of above tree.

Q. Create BST :

1. 50 ,25 ,75, 22,40,60,80,90,15,30

2. 10,3,15,22,6,45,65,23,78,34,5

Algorithm recInsert (root <tree ptr>, new<pointer>)

Pre: root points to root of the tree

Return : new node inserted in tree

1. if (root == NULL)
    1. root = new
    2. root->left = NULL
    3. root->right = NULL
    4. return
2. else
    1. if (new->data < root->data)
        1. recInsert (root->left, new)
    2. else
        1. recInsert (root->right, new)

Marwadi University
Marwadi Chandarana Group

Deletion of node from BST

Deletion

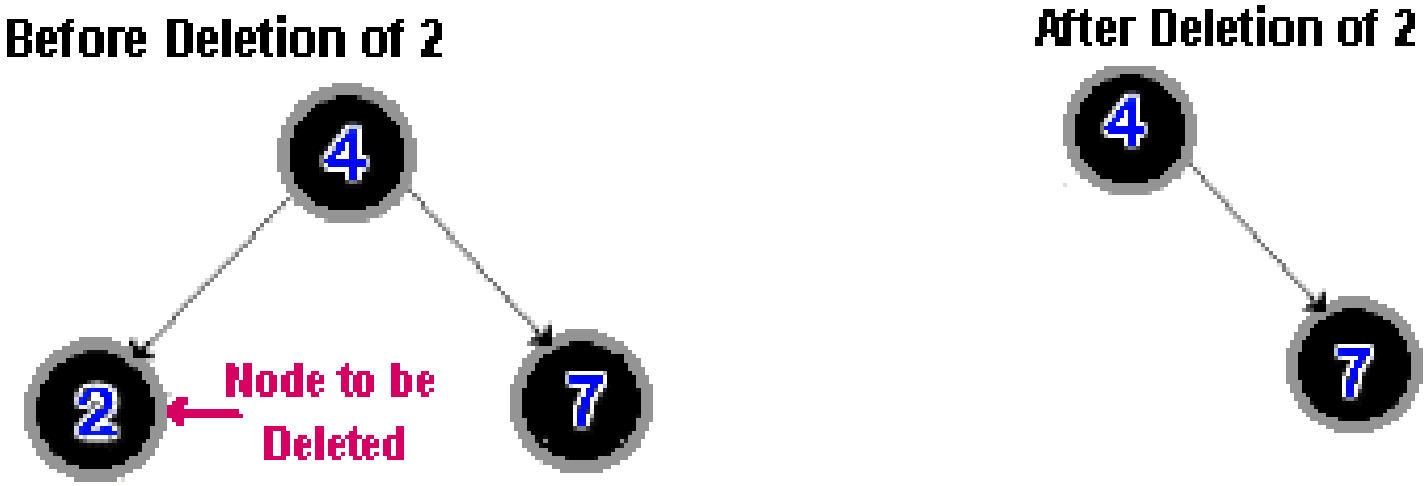
- Deleting a leaf
  - Simply remove it from the tree.
- Deleting a node with one child
  - Delete it and replace it with its child.
- Deleting a node with two children
  - Suppose the node to be deleted is called *N*. Replace node N with either.
    - In-order successor (the left-most child of the right subtree) OR (smallest value of right sub-tree).

    or

    - In-order predecessor (the right-most child of the left subtree). OR (Largest value of left sub-tree)

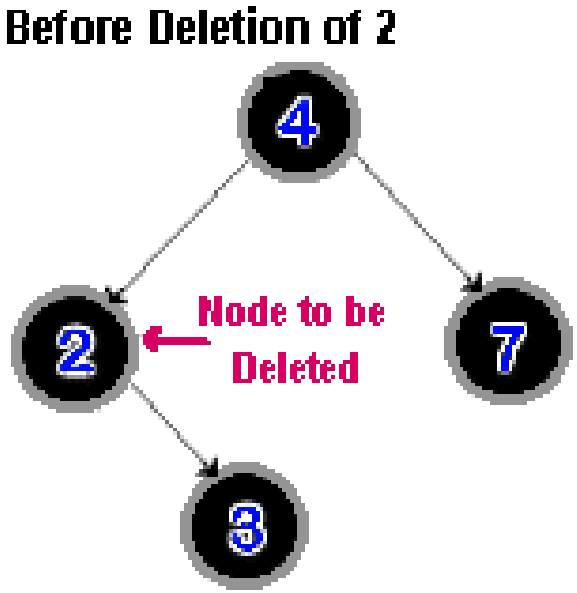
1. The node to be deleted has no children.

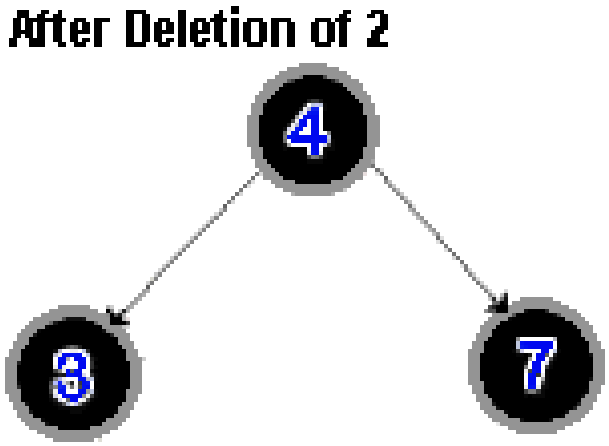
   In this case the node may simply be deleted from the tree.



2. The node has one child.

   The child node is appended to its grandparent.

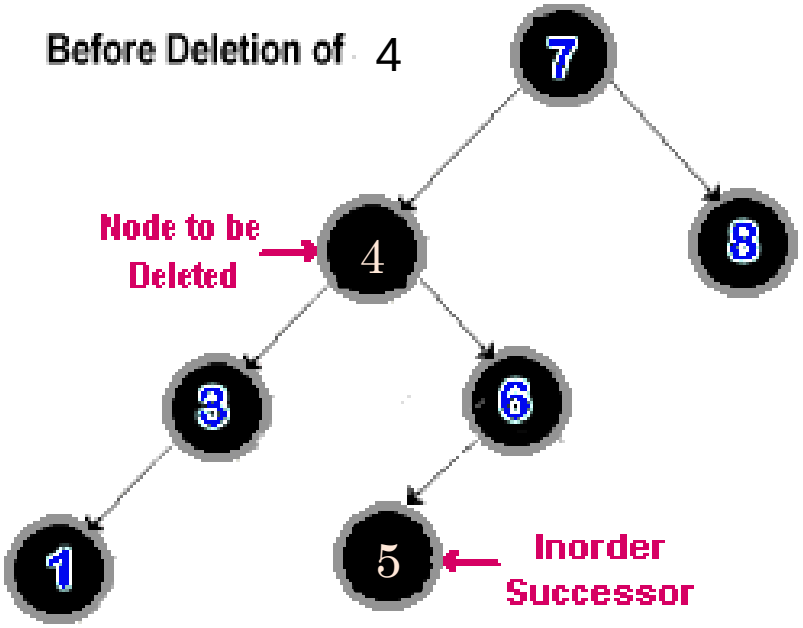


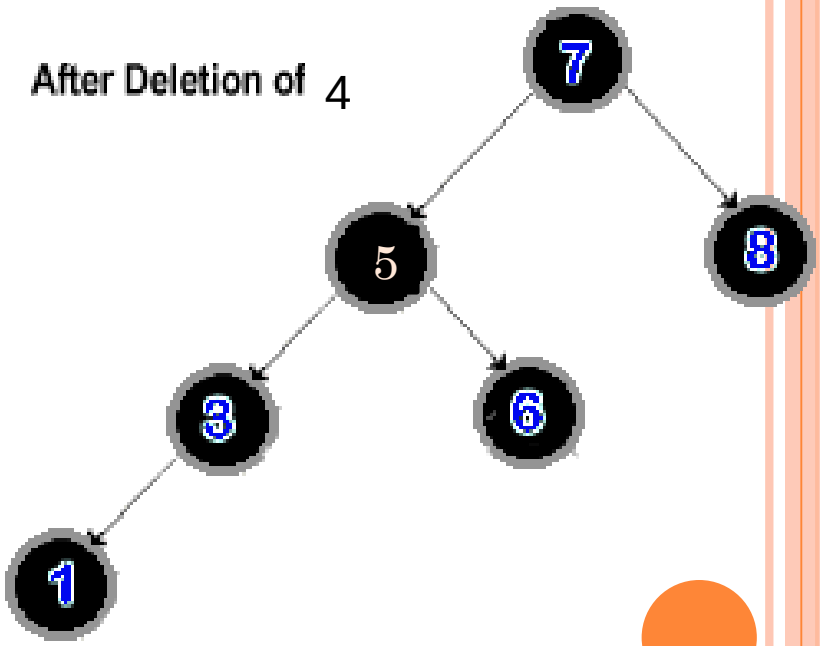


3. The node to be deleted has two children.

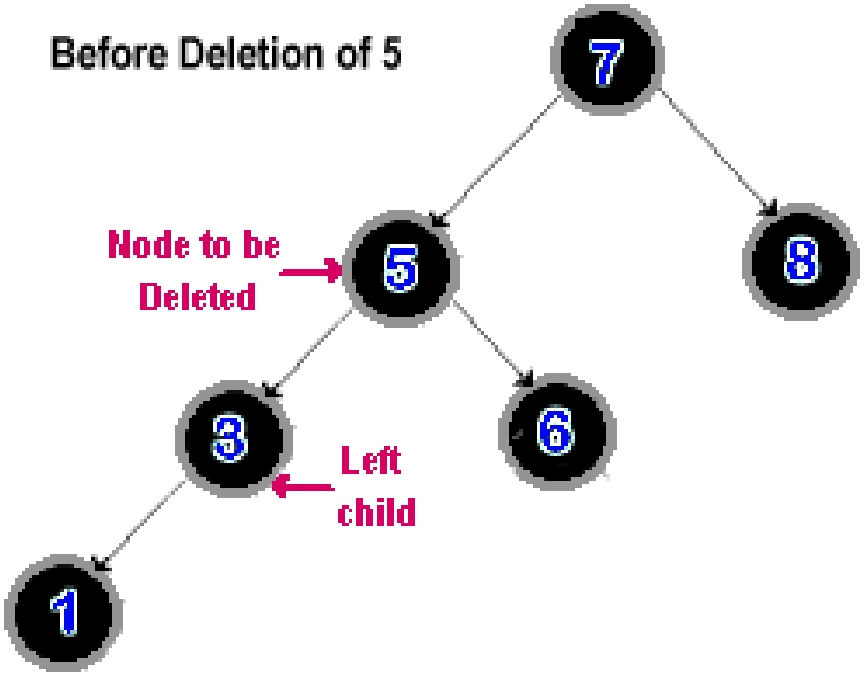
   This case is much more complex than the previous two, because the order of the binary search tree must be kept intact. The algorithm must determine which node to use in place of the node to be deleted:

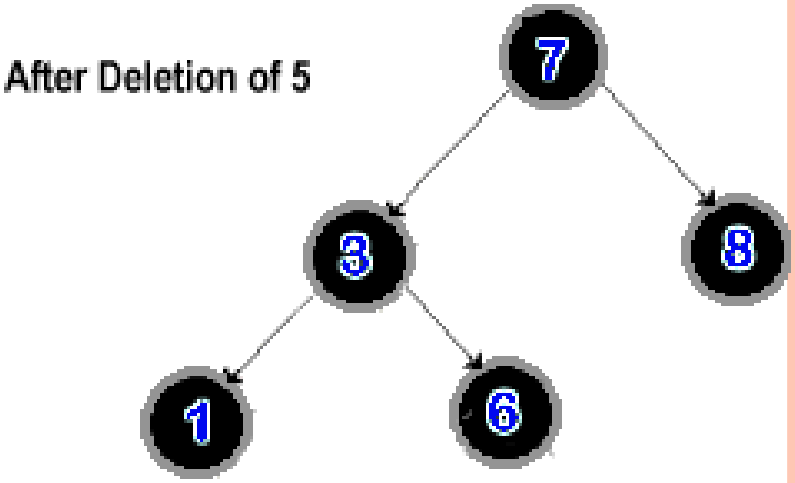
a. Replace node to be deleted with its inorder successor.



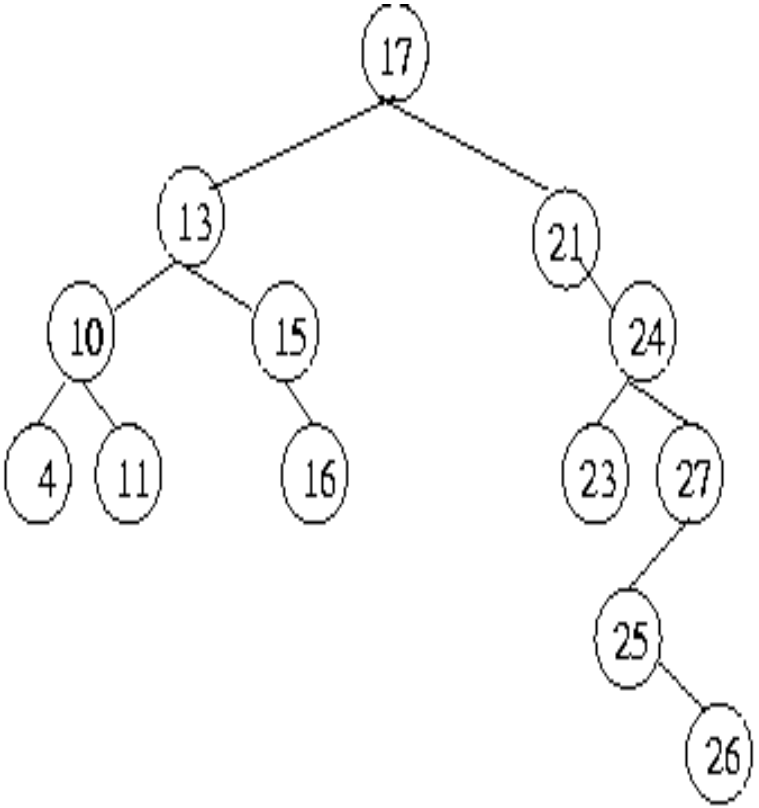


b. Replace node to be deleted with its inorder predecessor.

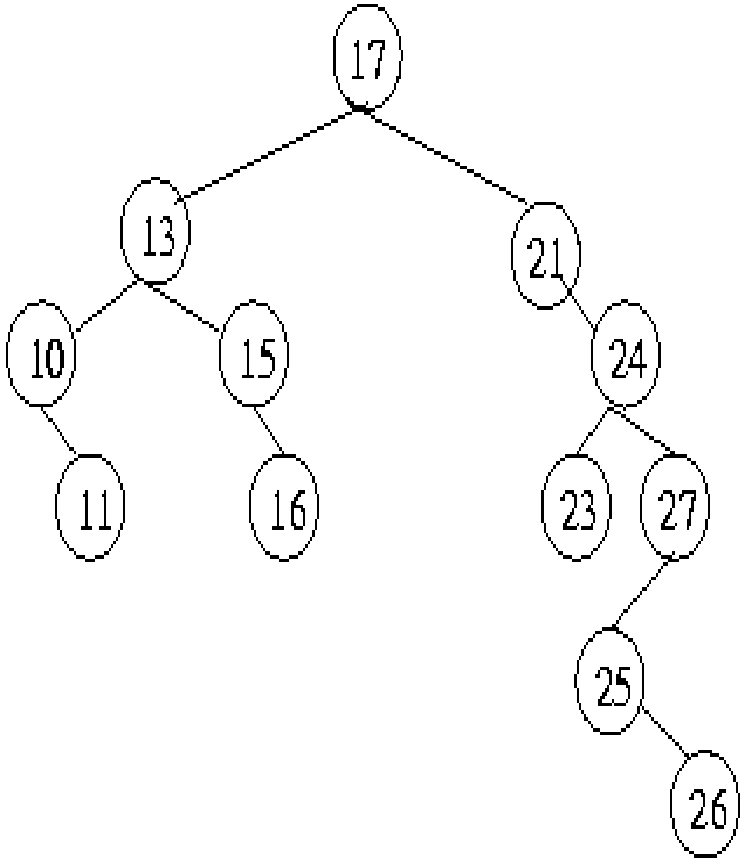


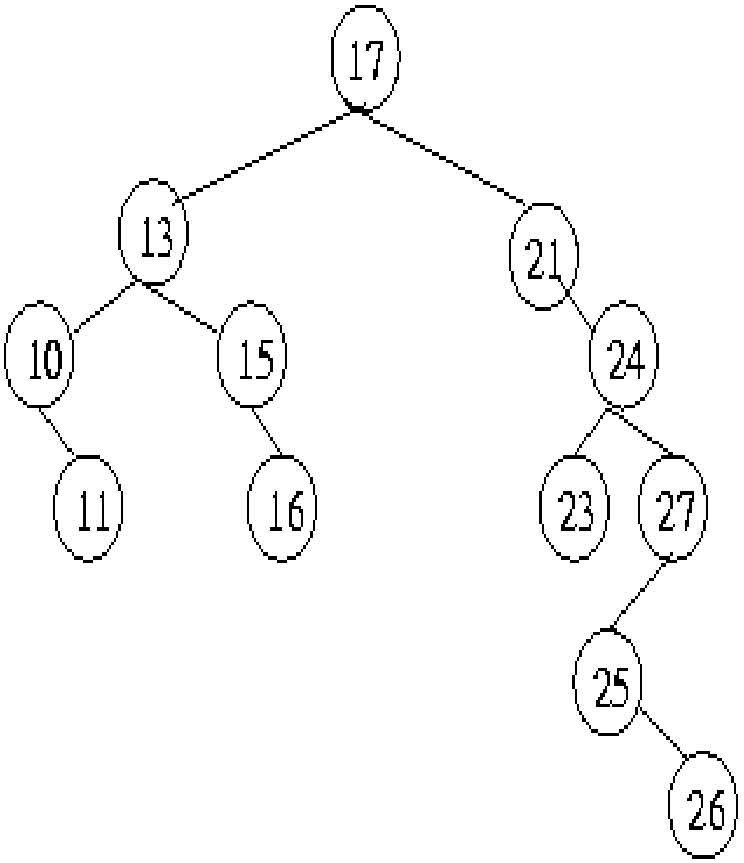


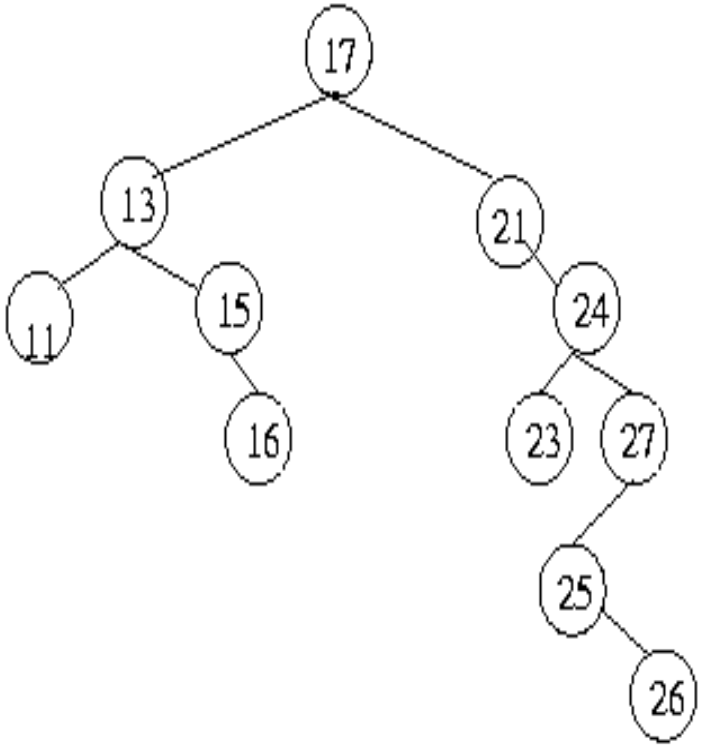
Deletion Exercise



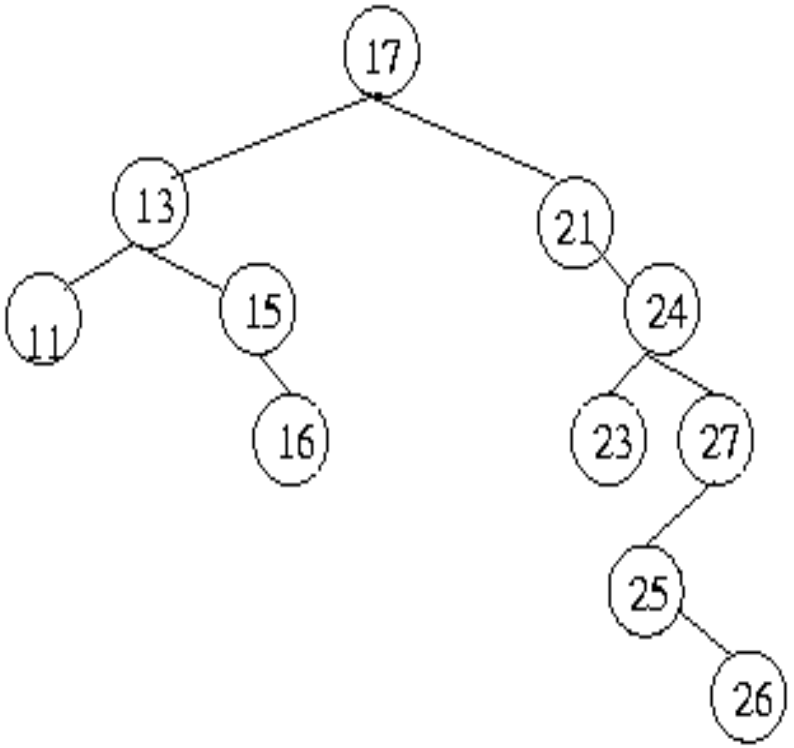
Delete 4

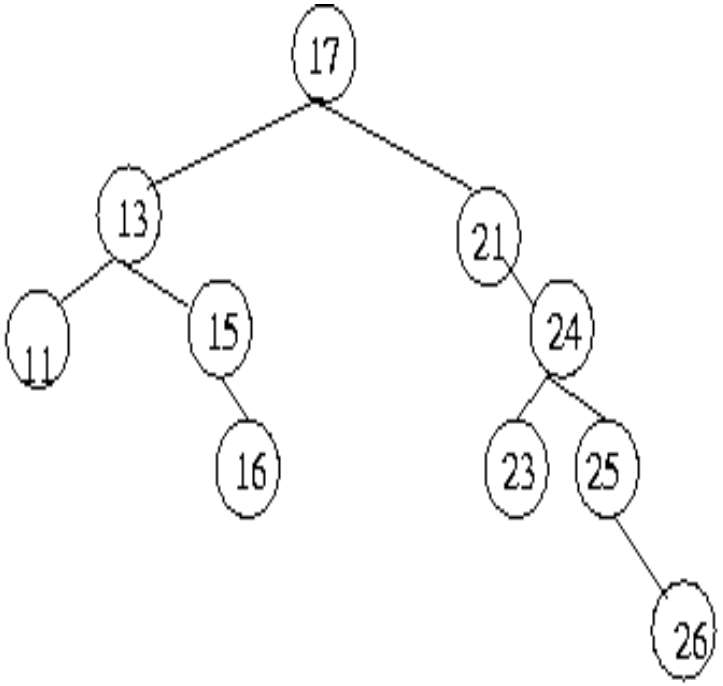




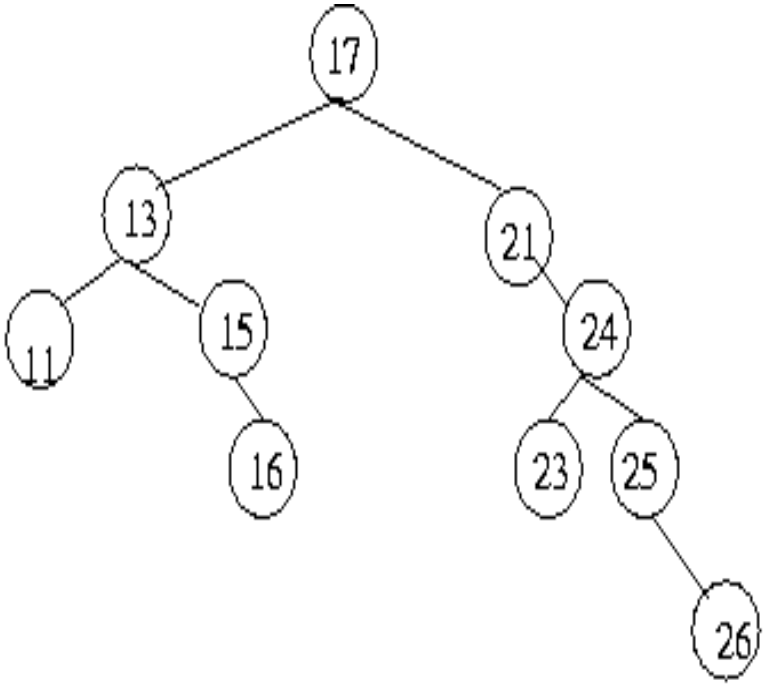


Delete 10

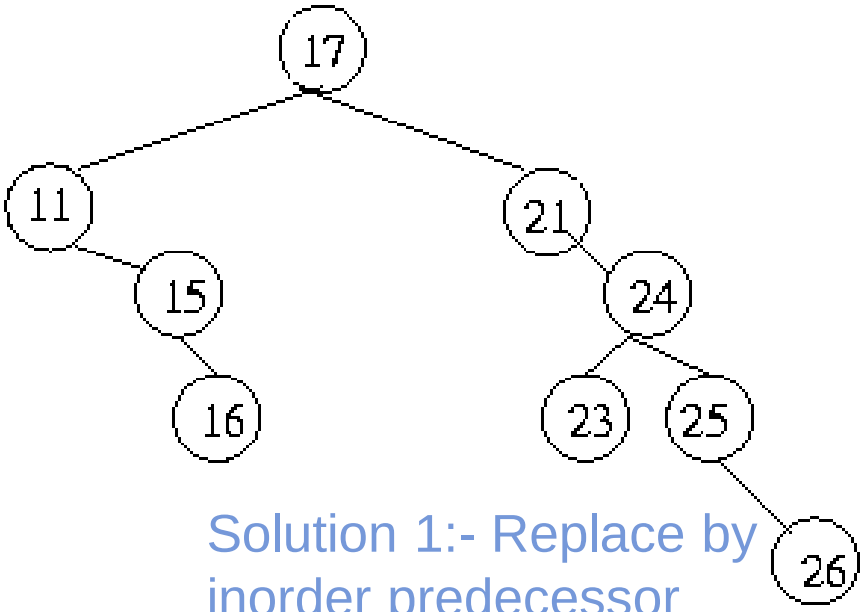




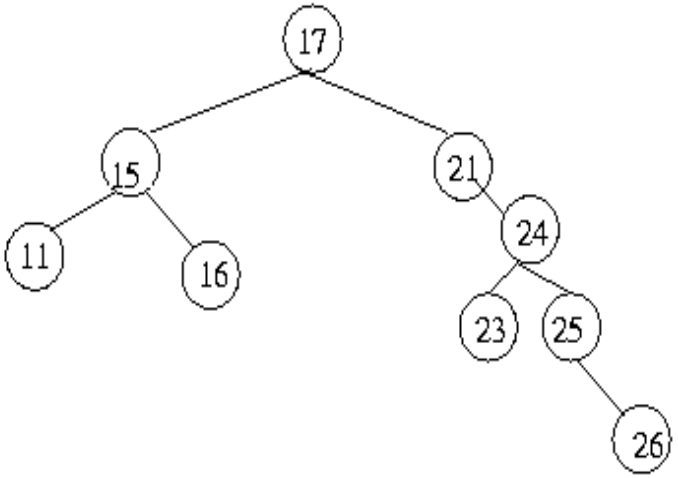
Delete 27



Delete 13



Solution 1:- Replace by inorder predecessor



Solution 2 :- Replace by inorder successor

Algorithm delete (root <tree ptr>, key<key value>)
Pre: root points to root of the tree, key is data to be deleted
Return : Deletes specified node, if found

```
1.    if (root == NULL)
      1.    return false
2.    if (key < root->data)
      1.    return delete (root->left, key)
3.    else if (key > root->data)
          1.    return delete (root->right, key)
4.    else
      1.    if (root->left == NULL)          //if node has only right child    //or node has no child
                1.    p = root
                2.    root = root->right
                3.    free p
                4.    return true
      2.    else if (root->right == NULL)    //if node has only left child
                1.    p = root
                2.    root = root->left
                3.    free p
                4.    return true
      3.    else     // find inorder predecessor, replace node to be deleted with inorder predecessor value
            1.    p = root->left
            2.    while ( p->right <> NULL)
                  1.    p = p->right
            3.    root->data = p->data
            4.    return delete (root->left, p->data)      //delete inorder predecessor
```

// for inorder successor

1. p = root->right // p = root->left
2. while (p->left <> Null) // while (p->right <> Null)
    1. p = p->left // p = p->right
3. root->data = p->data
4. return delete (root->right, p->data) // delete(root->left, p->data)

Marwadi University
Marwadi Chandarana Group

Reconstruction of Binary Tree from Traversal

Question

A binary tree has 10 nodes. The inorder and preorder traversal are shown below.

Inorder: A B C E D F J G I H

Preorder: J C B A D E F I G H

Show a step-wise reconstruction of the binary tree along with its postorder traversal.

Solution

Marwadi University
Marwadi Chandarana Group

Inorder: A B C E D F J G I H
Preorder: J C B A D E F I G H

Preorder(tree) = root | preorder(left ST) | preorder(right ST)
Inorder(tree) = Inorder(left ST) | root | Inorder(right ST)
Postorder(tree) = Postorder(left ST) | postorder(right ST) | root

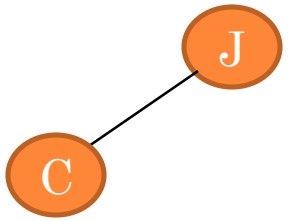
From preorder,
Root: J
Left subtree: A B C E D F
Right subtree: G I H

Preorder making subset [Cleft, Bleft, Aleft, Dleft, Eleft, Fleft, Iright, Gright, Hright]

All lefts are before all right nodes.

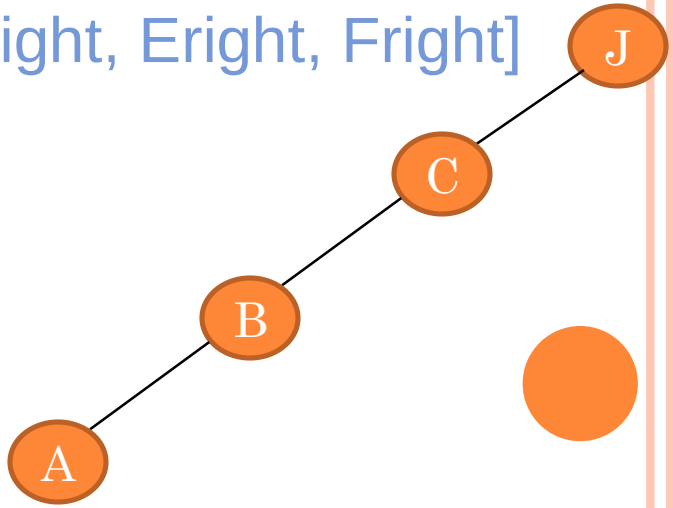
Solution

For left subtree of J:
Inorder: A B C E D F
Preorder: C B A D E F



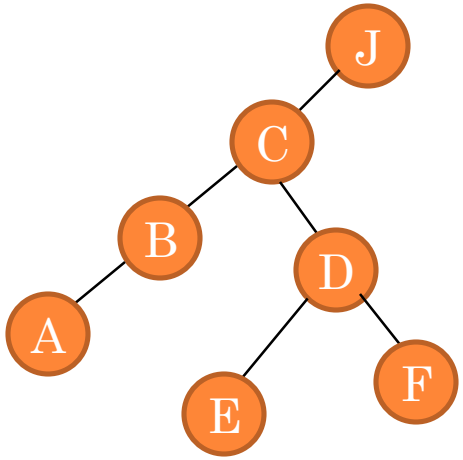
From preorder,
Root: C
Inorder finds
Left subtree: A B
Right subtree: E D F

Preorder making subset [Bleft, Aleft, Dright, Eright, Fright]
All lefts are before all right nodes.
For left subtree of C:
Inorder: A B
Preorder: B A
Root: B
[A] B i.e. A is at the left of B



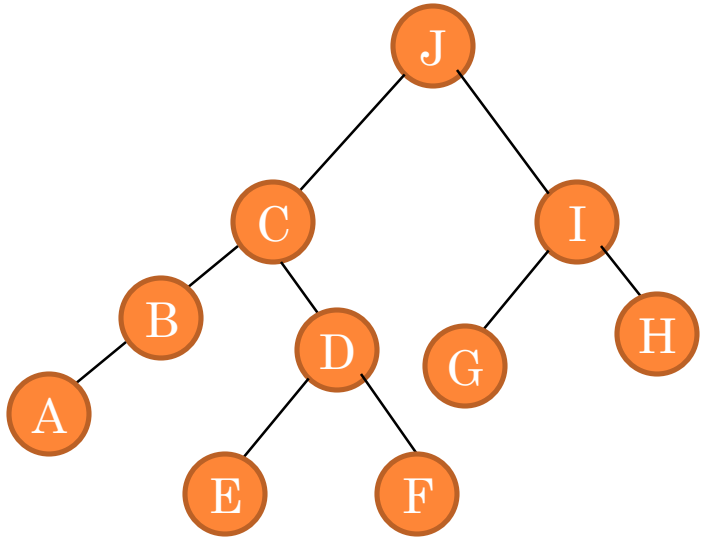
Solution

For right subtree of C:
Inorder: E D F
Preorder: D E F
Root: D
Preorder making subset [Eleft, Fright]
All lefts are before all right nodes.
[E] D [F]



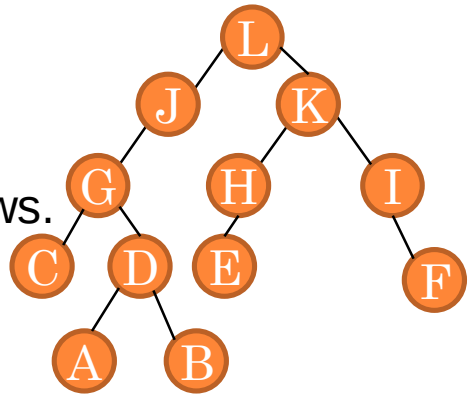
For right subtree of J:
Inorder: G I H
Preorder: I G H
Root: I
Preorder making subset [Gleft, Hright]
All lefts are before all right nodes.

[G] I [H]



EXERCISE

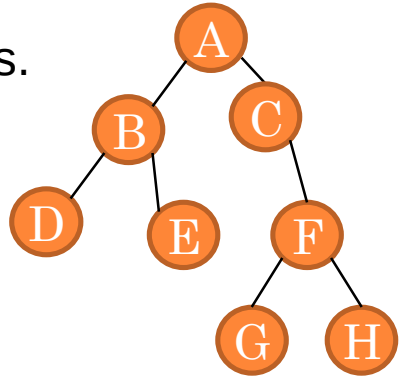
1. The inorder and preorder traversal of binary tree are as follows.



Inorder: C G A D B J L E H K I F
Preorder: L J G C D A B K H E I F

Draw binary tree and find its postorder traversal.

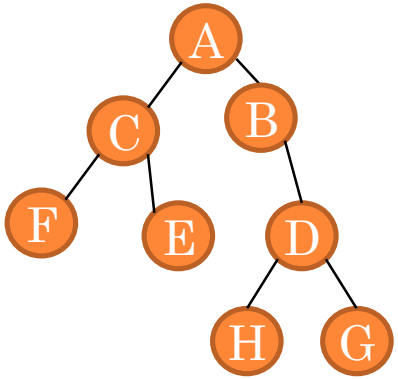
2. The inorder and preorder traversal of binary tree are as follows.



Inorder: D B E A C G F H
Preorder: A B D E C F G H

Draw binary tree and find its postorder traversal.

3. The inorder and postorder traversal of binary tree are as follows.



Inorder: F C E A B H D G
Postorder: F E C H G D B A

Draw binary tree and find its preorder traversal.

Marwadi University
Marwadi Chandarana Group

General Trees

General tree

- It is a tree in which each node can have an unlimited outdegree. Each node may have as many children as necessary.

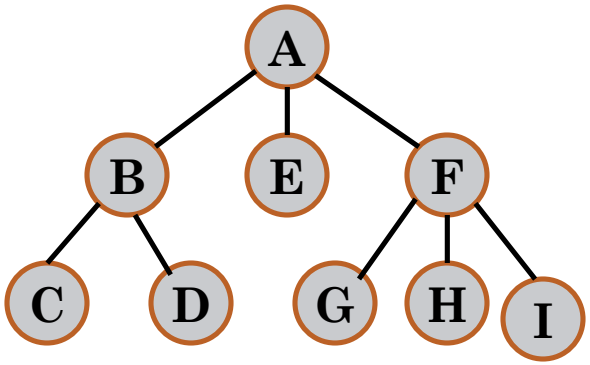
- It is easier to represent binary trees in program than representing general tree.

Steps to convert General tree into binary tree

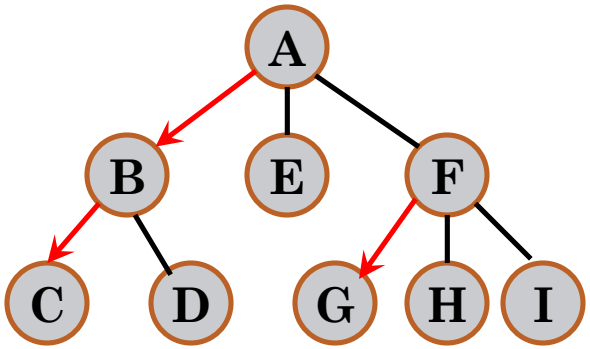
1. Identify branches from parents to their first or leftmost child. These branches become left pointers in the binary tree.

2. Connect siblings, starting with the leftmost child, using a branch for each sibling to its right sibling.

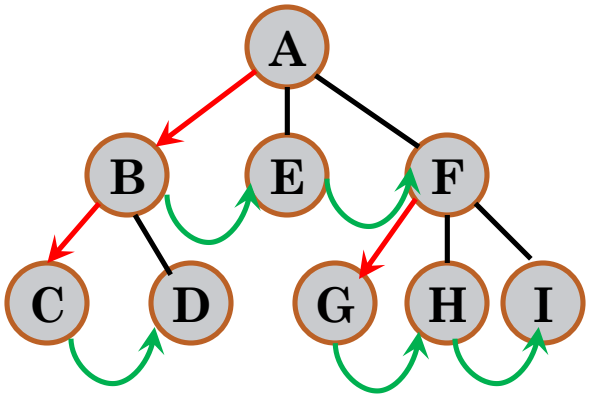
3. Remove all unneeded branches from the parent to its children.



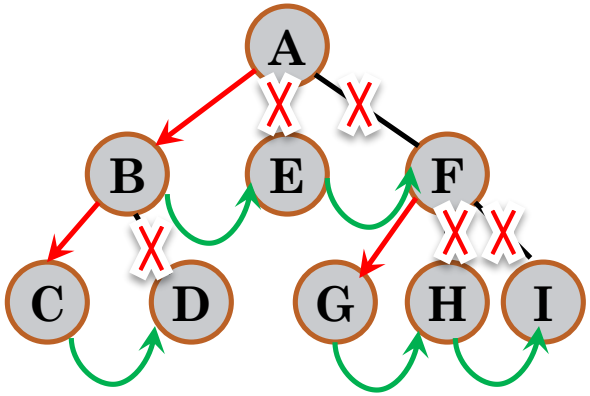
a) The general tree



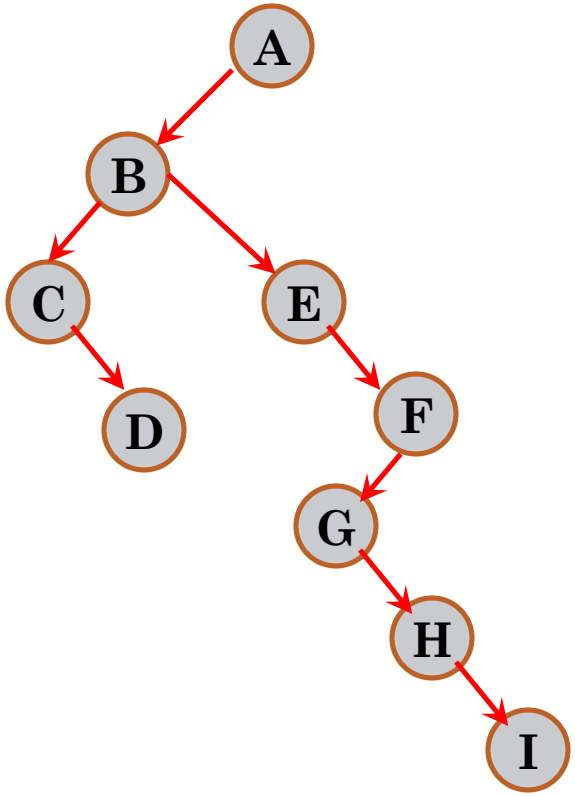
b) Identify leftmost children



c) Connect siblings



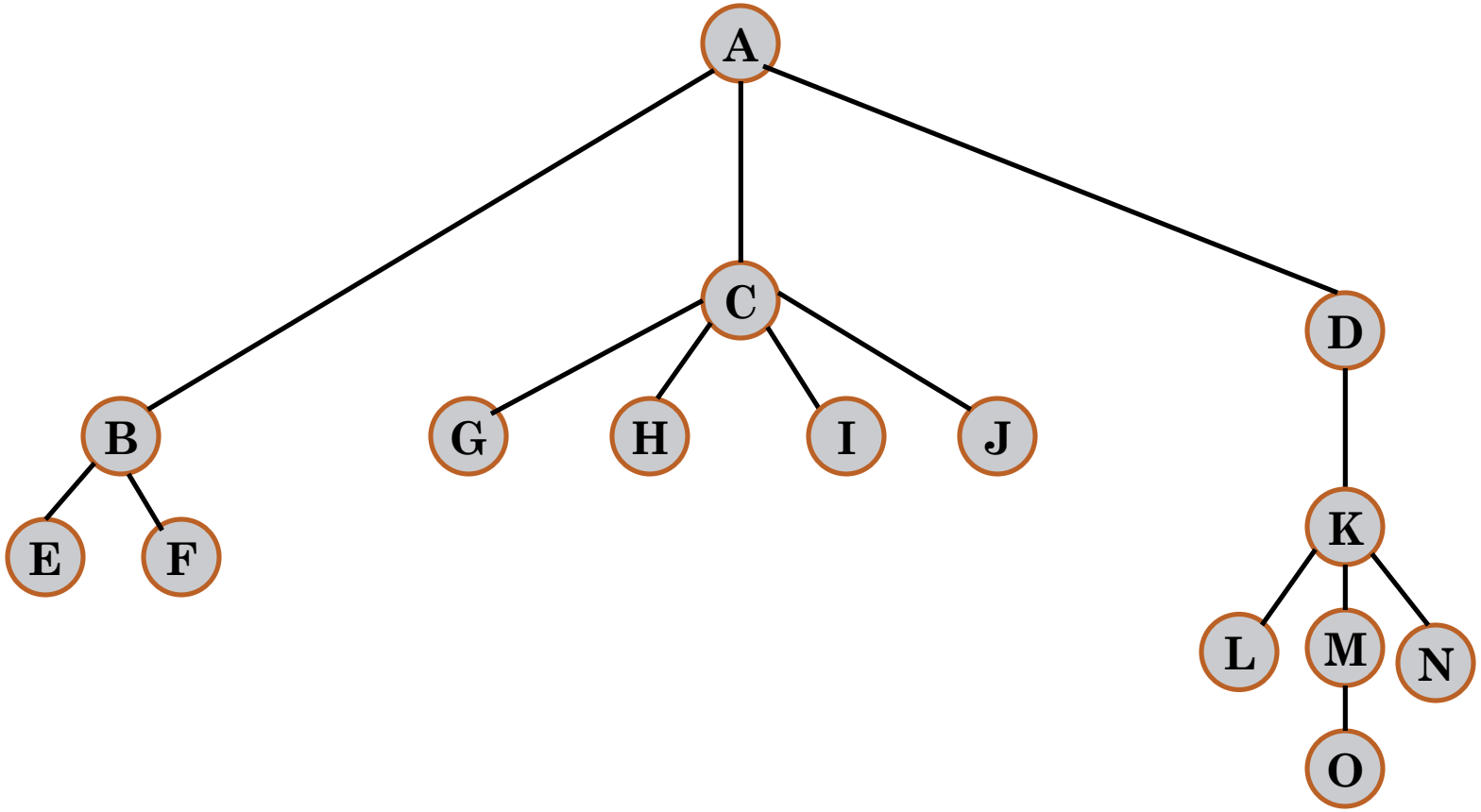
d) Delete unnecessary branches

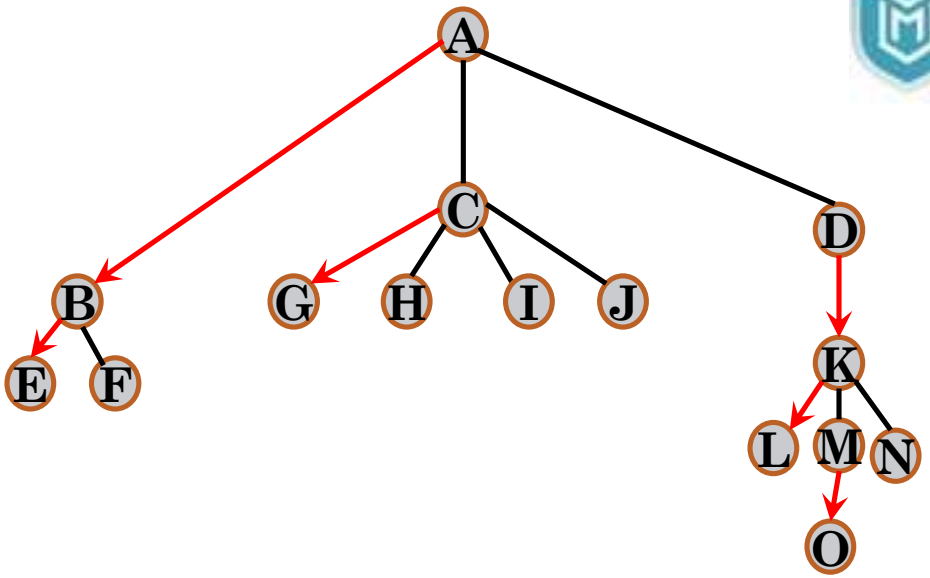


e) The resulting binary tree

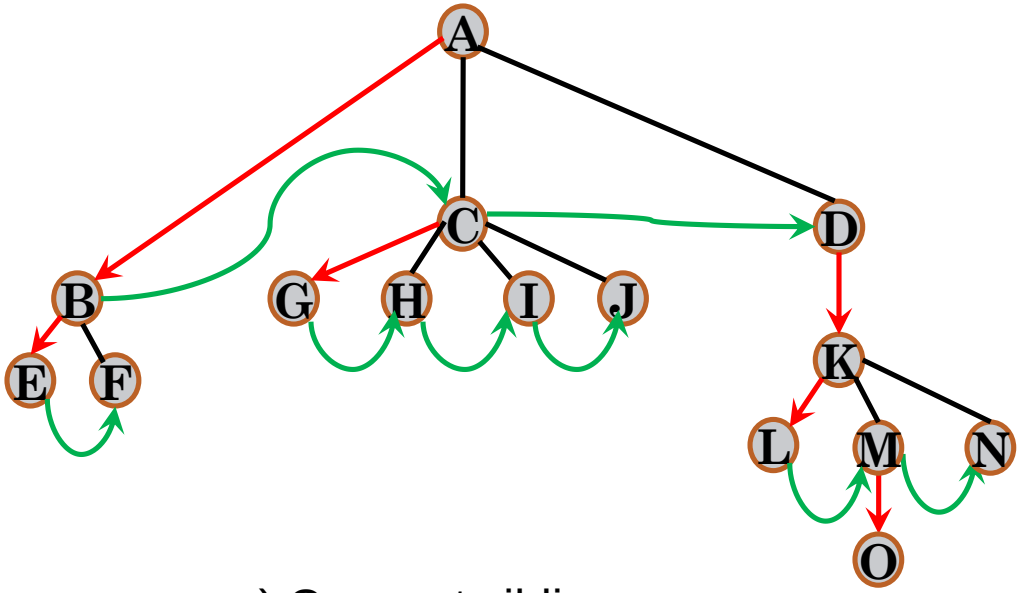
Exercise:

Convert following general tree into binary tree.

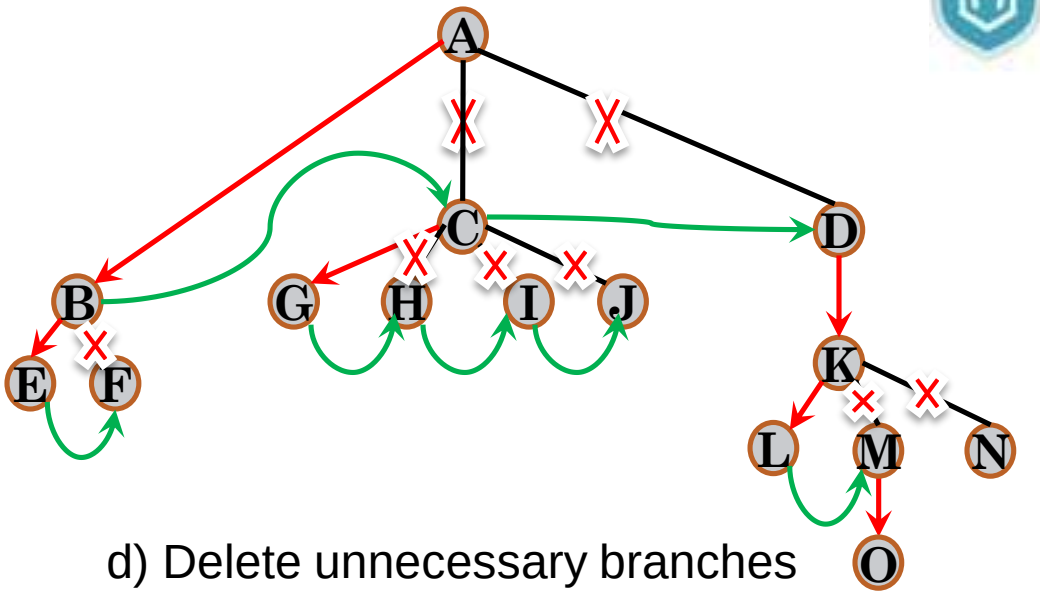




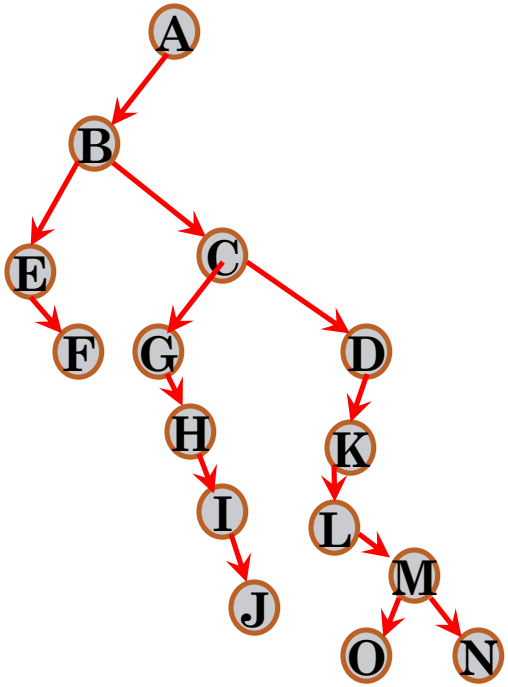
b) Identify leftmost children



c) Connect siblings



d) Delete unnecessary branches



e) The resulting binary tree

Marwadi University
Marwadi Chandarana Group

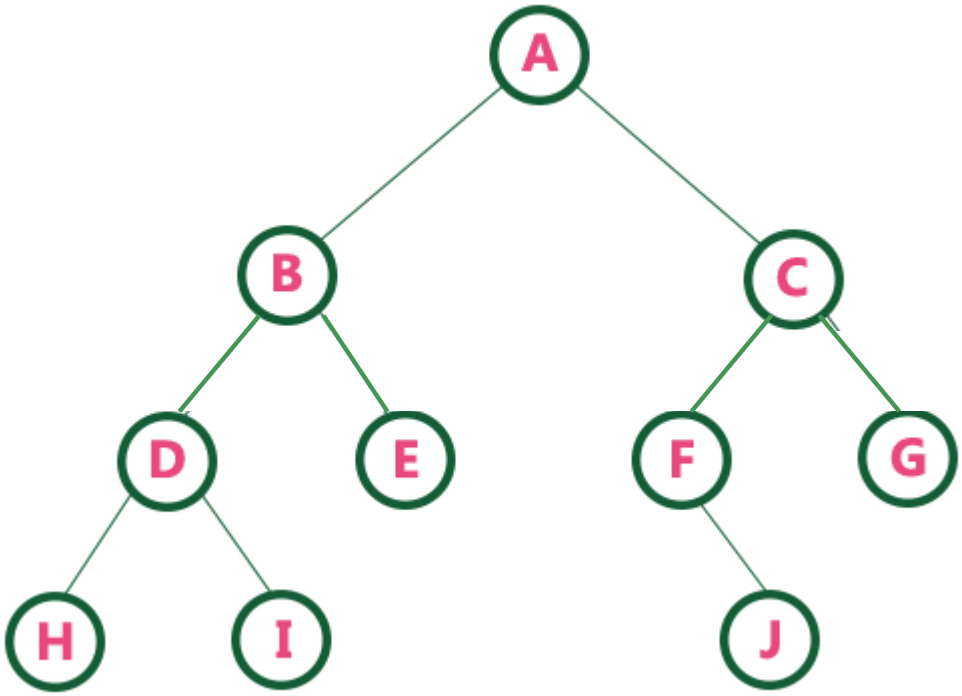
Threaded Binary Tree

- When a binary tree is represented using linked list representation, if any node is not having a child we use NULL pointer in that position.
- In any binary tree linked list representation, there are more number of NULL pointer than actual pointers.
- Generally, in any binary tree linked list representation, if there are **2N** number of reference fields, then **N+1** number of reference fields are filled with NULL (**N+1 are NULL out of 2N**).
- This NULL pointer does not play any role except indicating there is no link (no child).

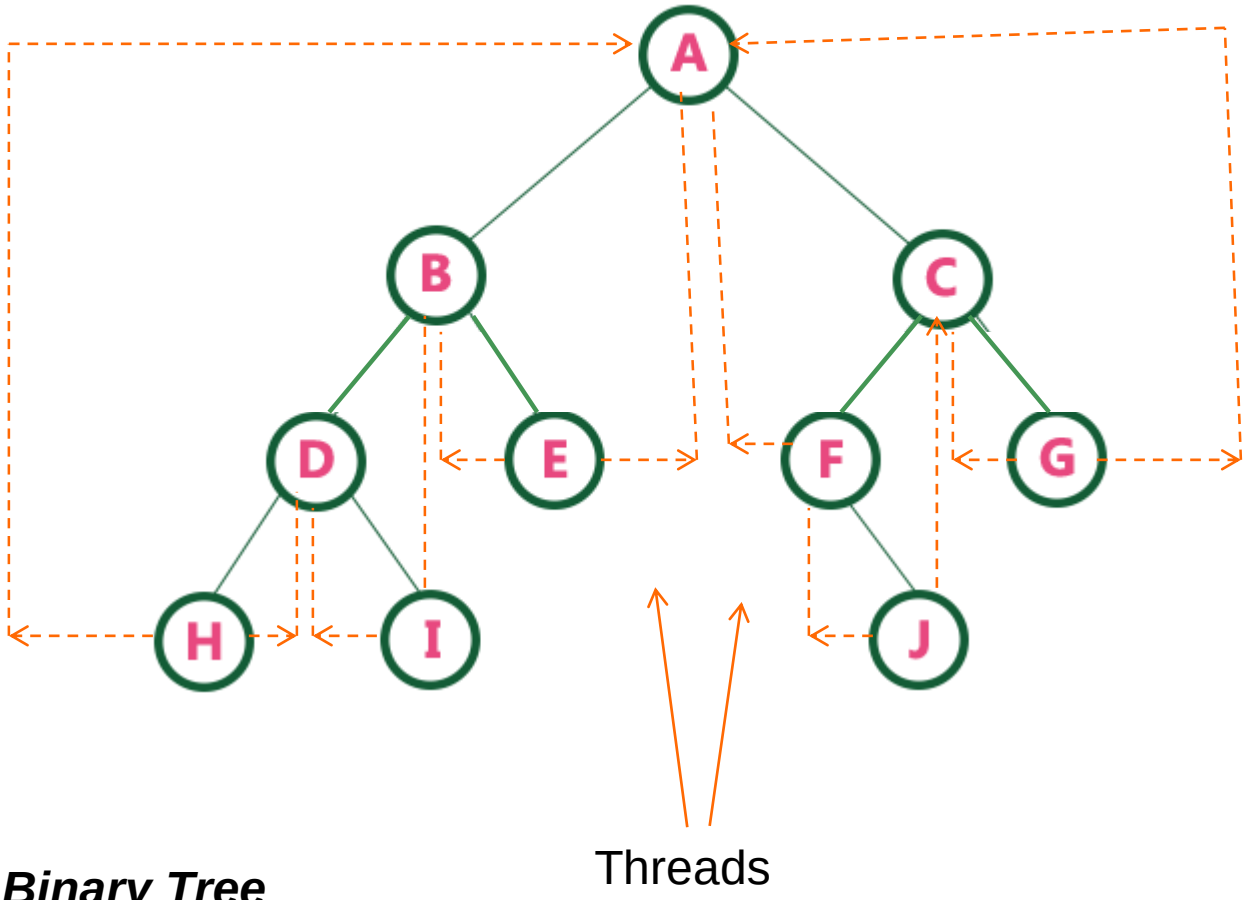
- **A. J. Perlis and C. Thornton** have proposed new binary tree called "***Threaded Binary Tree***", which make use of NULL pointer to improve its traversal processes.
- In threaded binary tree, NULL pointers are replaced by references to other nodes in the tree, called ***threads***.

- Threaded Binary Tree is also a binary tree in which all **left child pointers that are NULL** (in Linked list representation) points to **its in-order predecessor**, and all **right child pointers that are NULL** (in Linked list representation) points to **its in-order successor.**
- If there is no in-order predecessor or in-order successor, then it point to root node.
-

- Convert Following tree into threaded binary tree



- First find the in-order traversal of that tree…
  - **In-order traversal of above binary tree…**
  - **H - D - I - B - E - A - F - J - C - G**



Threaded Binary Tree

Marwadi University
Marwadi Chandarana Group

THANK YOU