Unit 4
Tree

Prof. Brijesh Vala, Assistant Professor
Computer Science & Engineering

Parul®
University

DIGITAL LEARNING CONTENT

Topic-1

Tree

Parul®
University

DIGITAL LEARNING CONTENT

Objectives

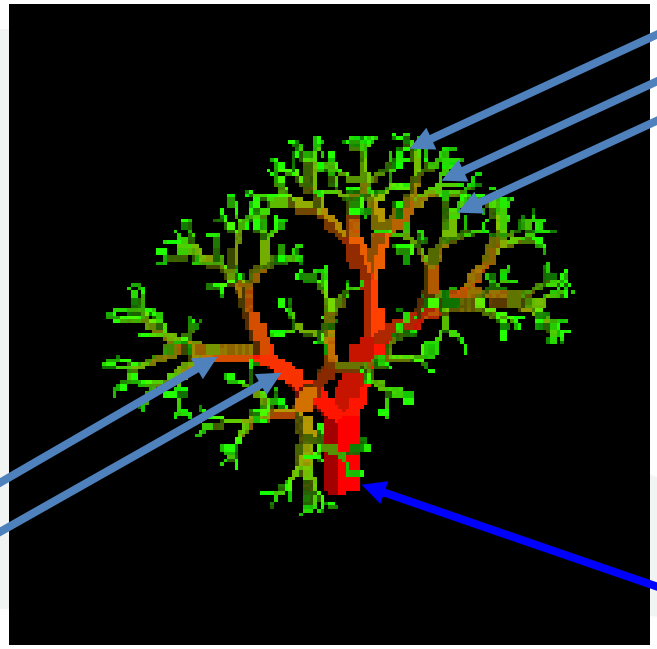
At the end of this unit, you will be able to understand the:

- basic tree terminologies **(concepts)**
- Various types of trees
- Various operation perform on binary trees

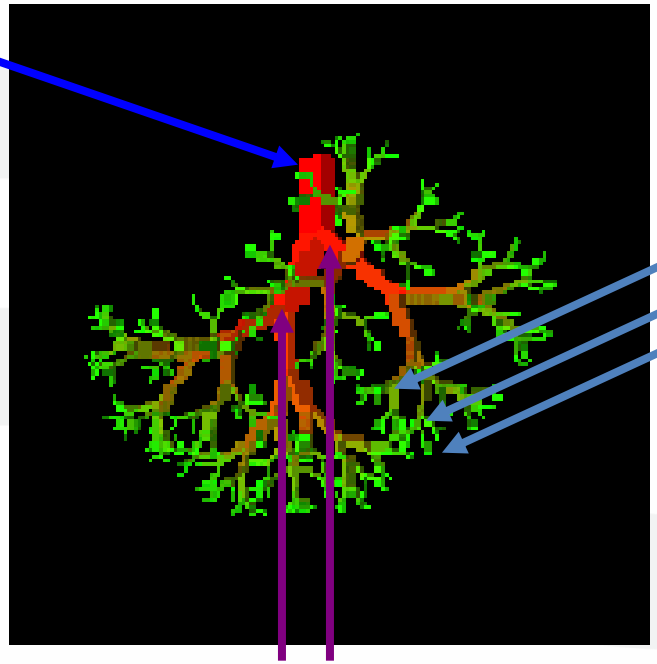
Parul®
University

DIGITAL LEARNING CONTENT

Nature View of a Tree



Computer Scientist's View



Basic Tree Concepts

- A **tree** is a non-linear data structure. A tree consists of finite set of elements (one or more), called nodes, and a finite set of directed lines called branches, that connect the nodes.
- Tree is an acyclic digraph.
- The elements are connected in such a way that there are no loops.
- There is only a unique path from source to destination in a tree.

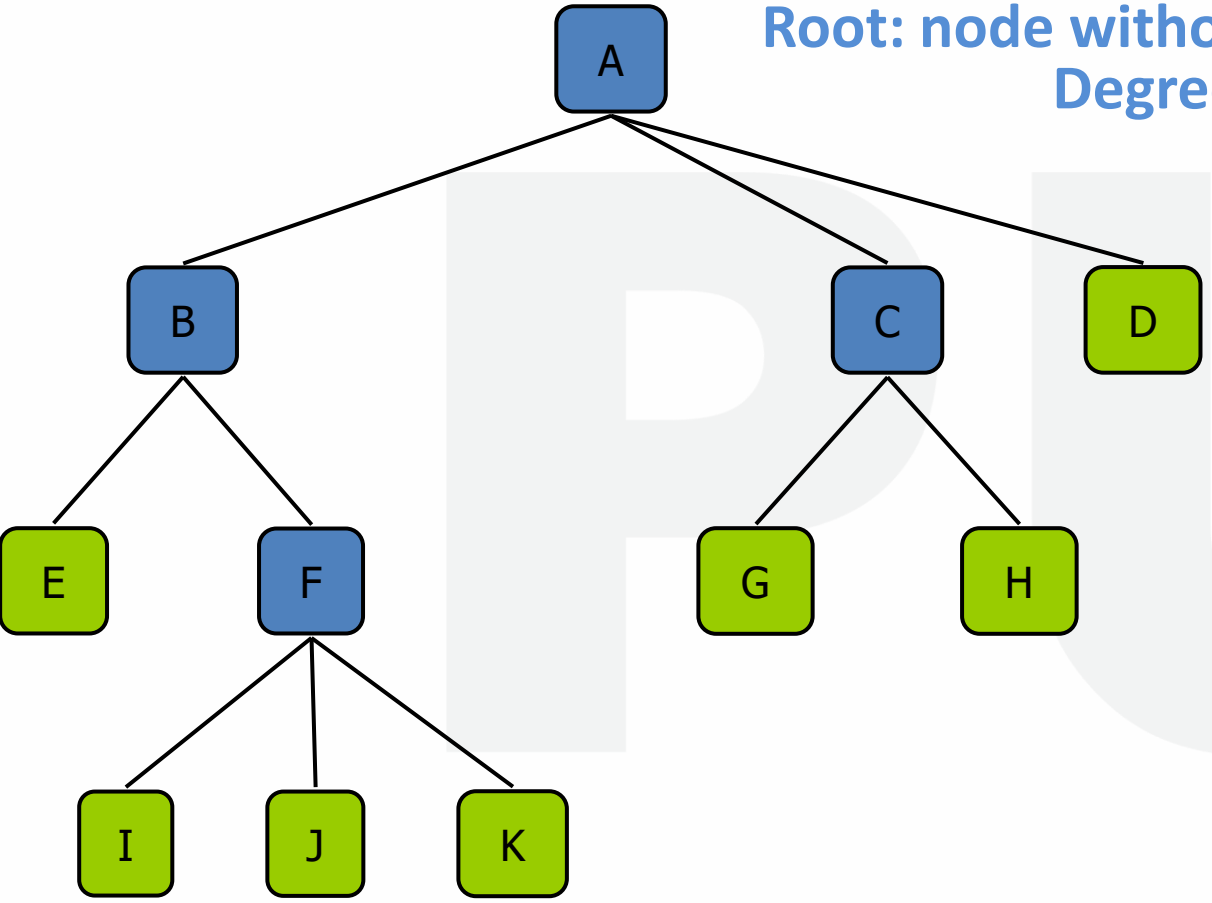
Basic Tree Concepts

- When the branch is directed toward the node, it is **indegree** branch.
- When the branch is directed away from the node, it is an **outdegree** branch.
- The sum of the indegree and outdegree branches is the **degree** of the node.
- If the tree is not empty, the first node is called the root.

Basic Tree Concepts

- **Root** node is the first node in the hierarchical arrangement. There can be only one root in the tree.
- **Parent** node is an immediate predecessor of a node.
- The immediate successors are called **child.**

Basic Tree Concepts



Root: node without any parent (A)
Degree(B)=3

Basic Tree Concepts

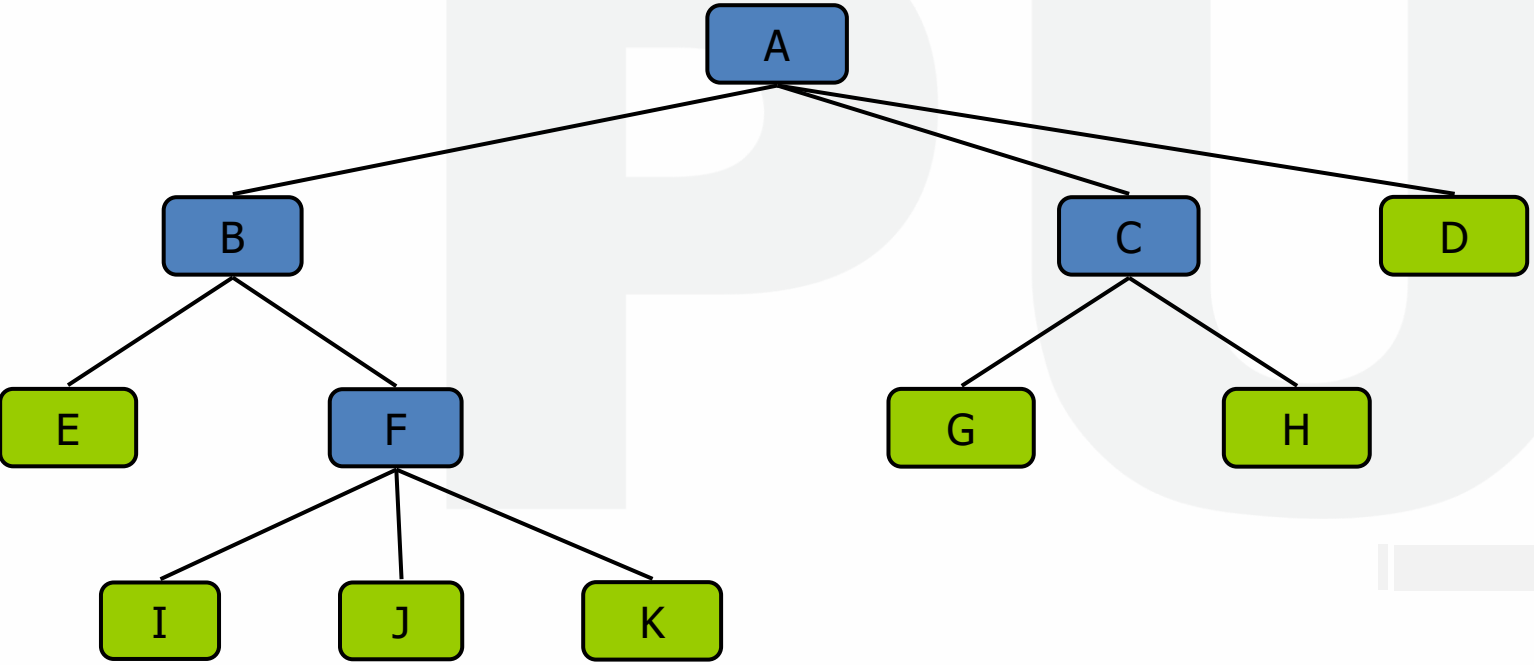
- The in-degree of the root is zero.
- With the exception of the root, all of the nodes in a tree must have an in-degree of exactly one; that is, they may have only one predecessor.
- All nodes in the tree can have out-degree of zero, one, or more.

Basic Tree Concepts

- A **leaf** is any node with an out-degree of zero, that is, a node with no successors. (a node without any further child node)
- A node that is not a root or a leaf is known as an **internal** node.
- A node is a parent if it has successor nodes; that is, if it has outdegree greater than zero.
- A node with a predecessor is called a child.

Basic Tree Concepts

- leaf: i,j,k,e,g,h,d
- Internal: b,c,f
- Parent: root+any internal node (A,B,C,F)
- Child: except root node A

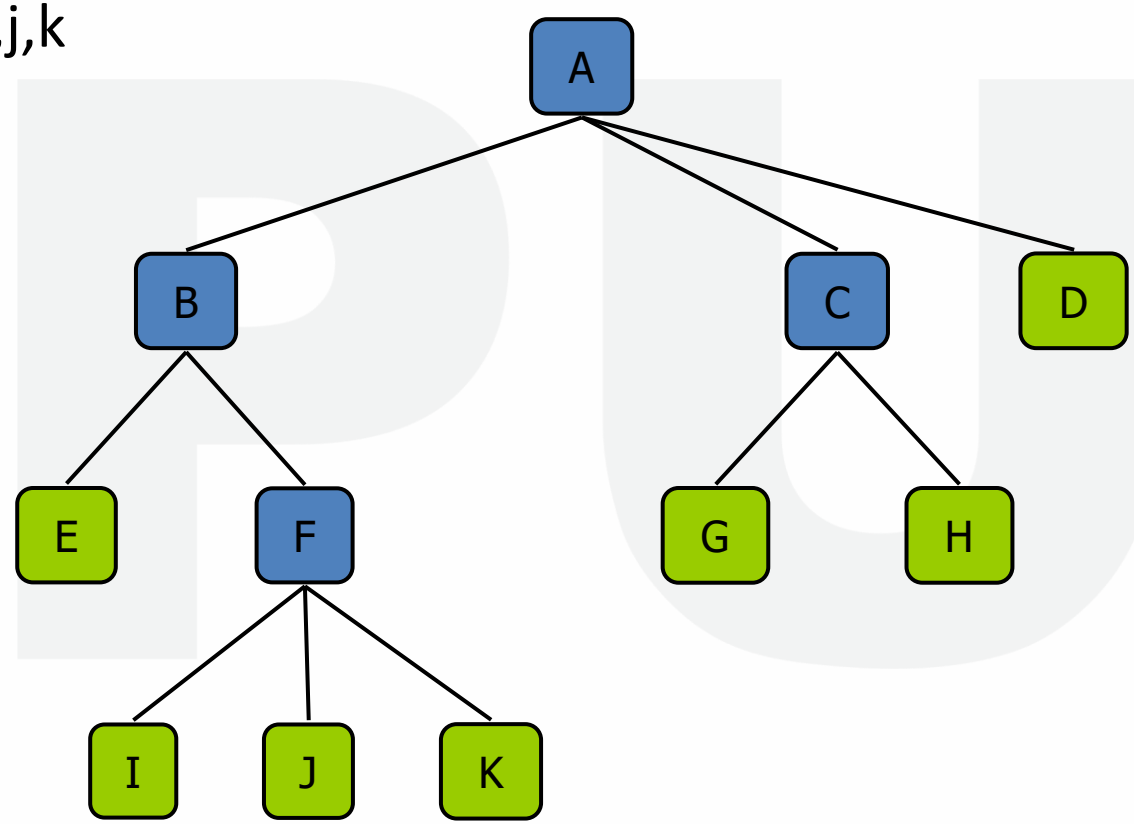


Basic Tree Concepts

- Two or more nodes with the same parents are called **siblings.**
- An **ancestor** is any node in the path from the root to the node.
- A **descendant** is any node in the path below the parent node; that is, all nodes in the paths from a given node to a leaf are descendants of that node.

Basic Tree Concepts

Sibling: c,d
Ancestor(f):b,a
Descendant(f):i,j,k



Parul®
University

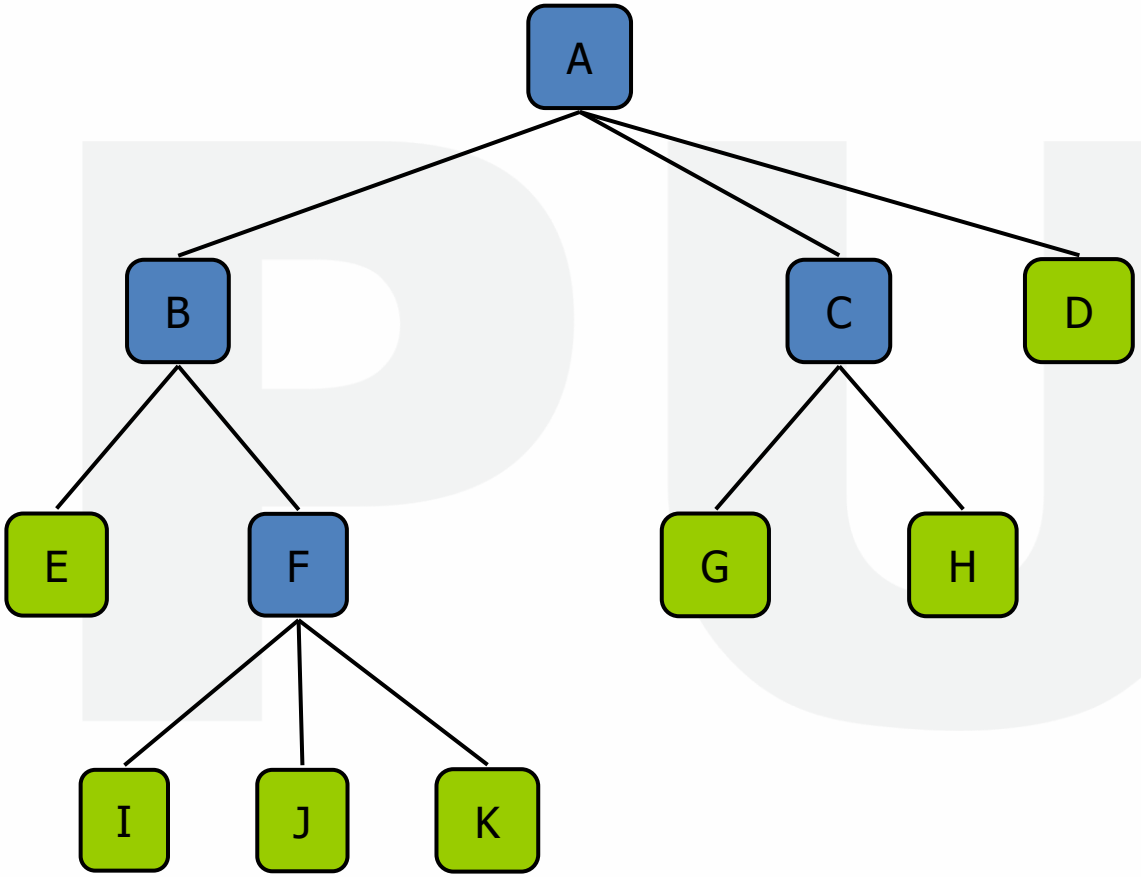
DIGITAL LEARNING CONTENT

Basic Tree Concepts

- A path is a sequence of nodes in which each node is adjacent to the next node. It is number of successive edges from source node to destination node.
- The level of a node is its distance from the root. The root is at level 0, its children are at level 1, etc. …

Basic Tree Concepts

Path(B,K):BFK
LEVEL(G):2



Parul®
University

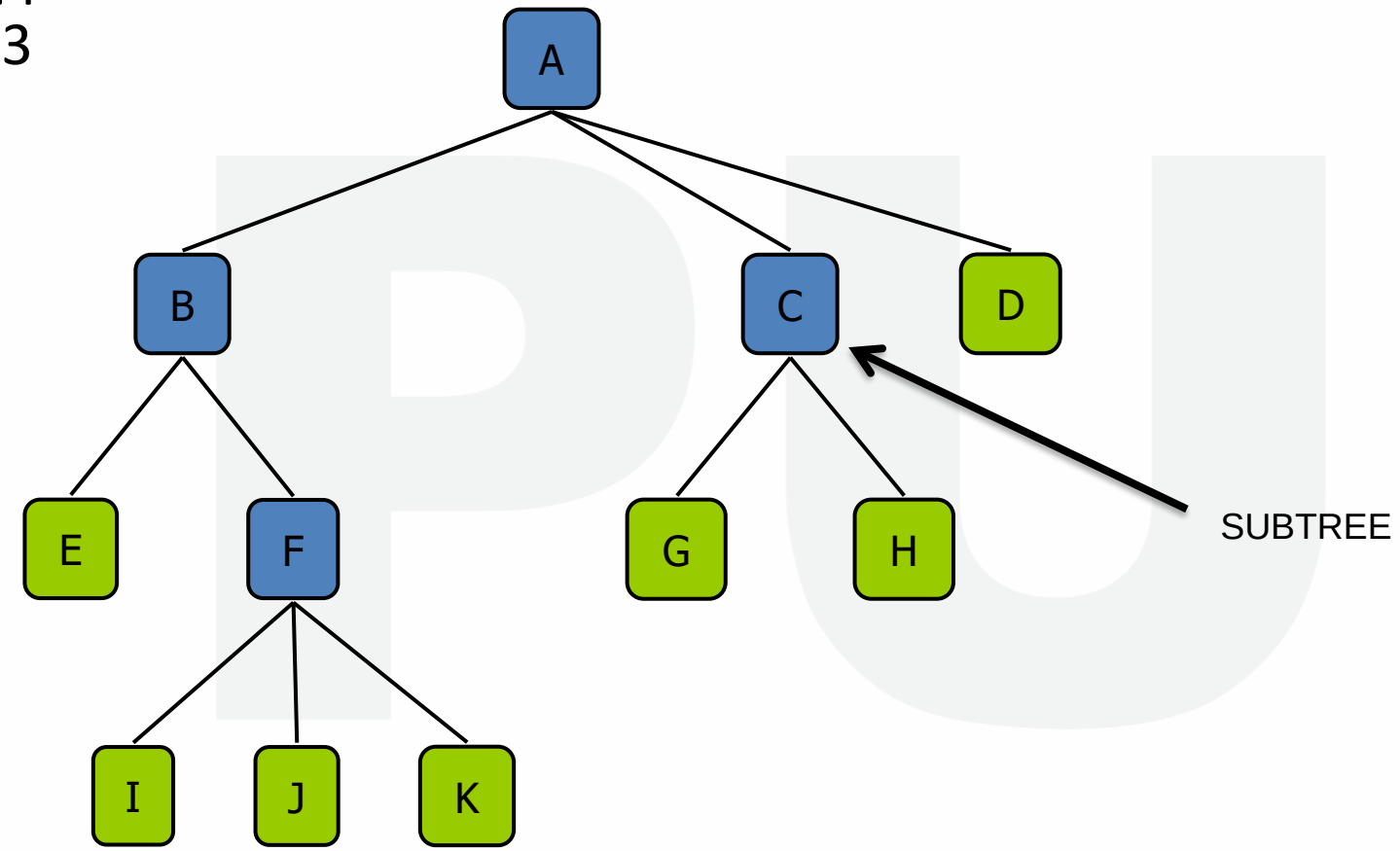
DIGITAL LEARNING CONTENT

Basic Tree Concepts

- The **height** of the tree is the total number of levels in the tree.
- The **depth** of the tree is last(max) level number of the tree.
- A **sub-tree** is any connected structure below the root. The first node in the subtree is known as the root of the subtree.

Basic Tree Concepts

HEIGHT:4
DEPTH: 3



Parul®
University

Different types of Trees

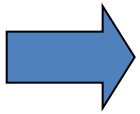
- Binary tree
- Threaded Binary Tree
- Binary Search Tree
- AVL Tree
- B Tree
- B+ Tree
- 2-3 Tree

Binary Trees

- A **binary tree** is a tree in which no node can have more than two children (subtrees); the maximum outdegree for a node is two.
- In other words, a node can contain either zero, one, or two subtrees.
- These subtrees are designated as the **left subtree** and the **right subtree**.

Binary Trees

A null tree is a tree with no nodes



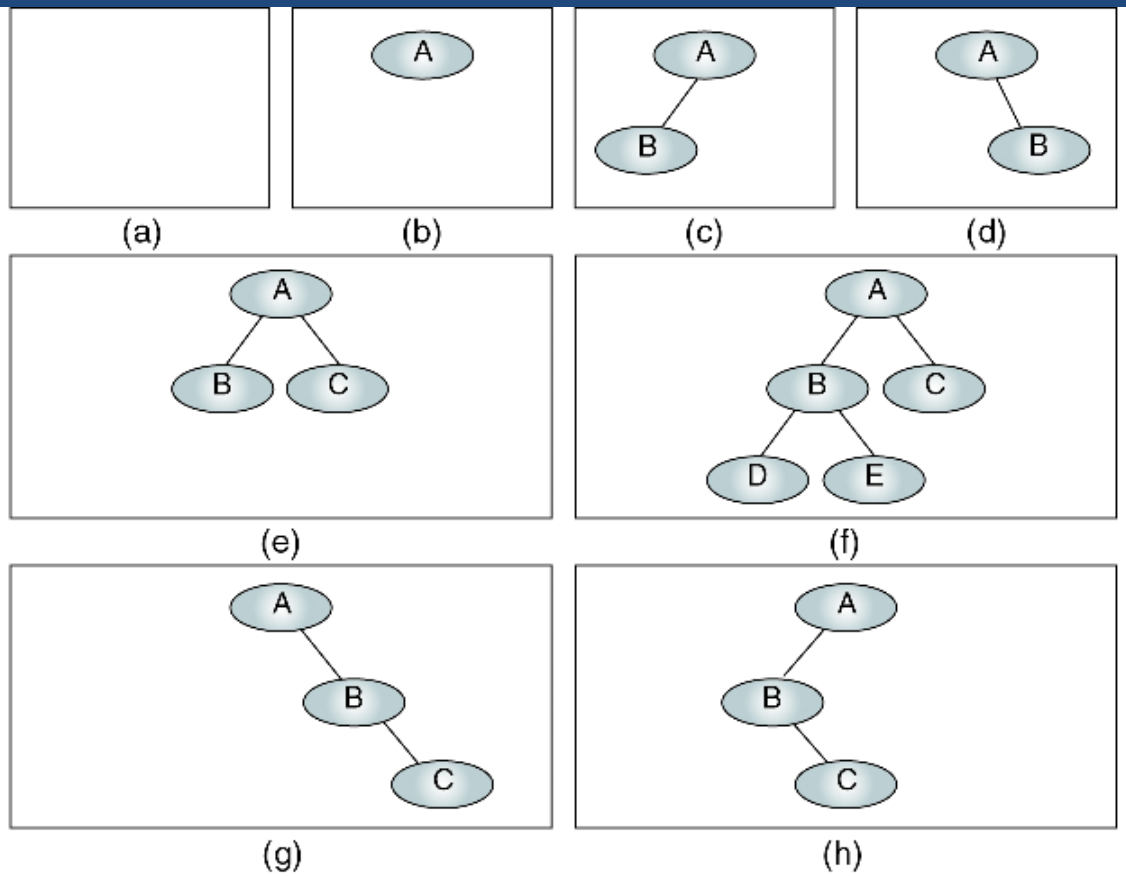


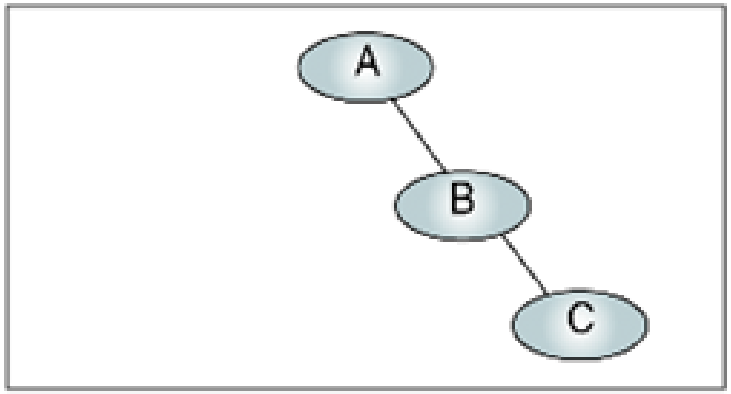
FIGURE 6-6 Collection of Binary Trees

Some Properties of Binary Trees

- The height of binary trees can be mathematically predicted
- Given that we need to store N nodes in a binary tree, the **maximum height** is

$$H_{max} = N$$

Height : 3
Total node : 3



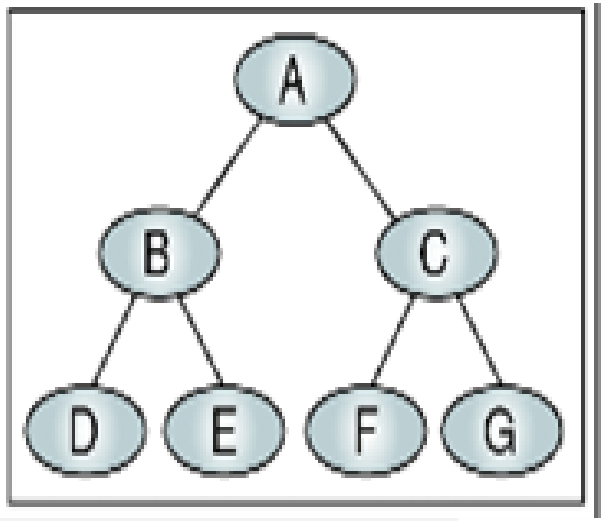
A tree with a maximum height is rare. It occurs when all of the nodes in the entire tree have only one successor.

Some Properties of Binary Trees

- The **minimum height** of a binary tree is determined as follows:

$$H_{\min} = \lceil \log_2 N \rceil + 1$$

Height : 3
Total node : 7

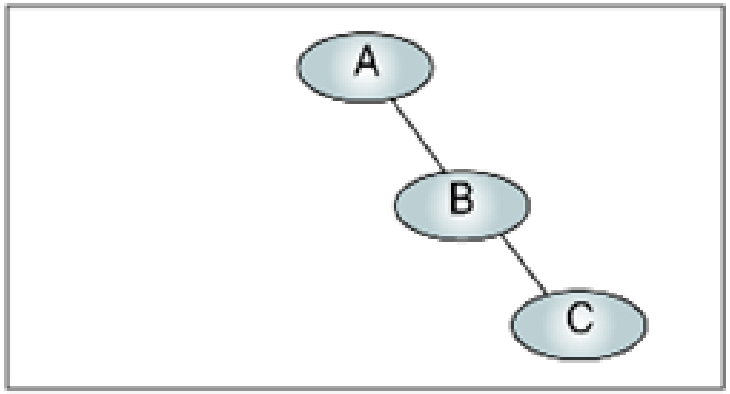


For instance, if there are three nodes to be stored in the binary tree (N=3) then $H_{\min}$=2.

Some Properties of Binary Trees

•Given a height of the binary tree, H, the **minimum number** of nodes in the tree is given as follows:

$$N_{\min} = H$$

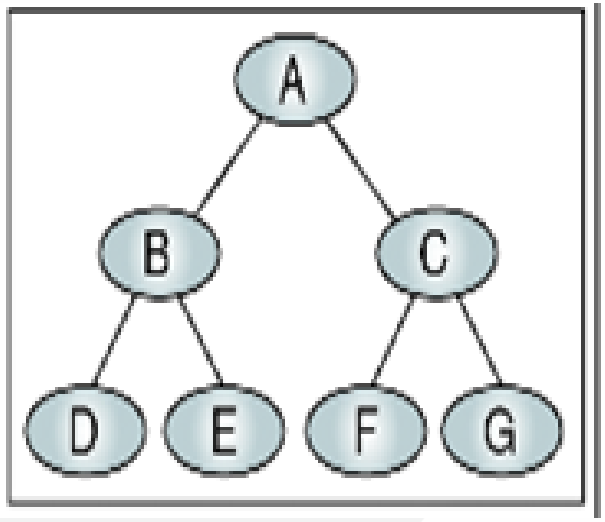


Some Properties of Binary Trees

•The formula for the maximum number of nodes is derived from the fact that each node can have only two child except leaf node.

$$N_{max} = 2^{H} - 1$$

Height : 3
Total nodes : 7



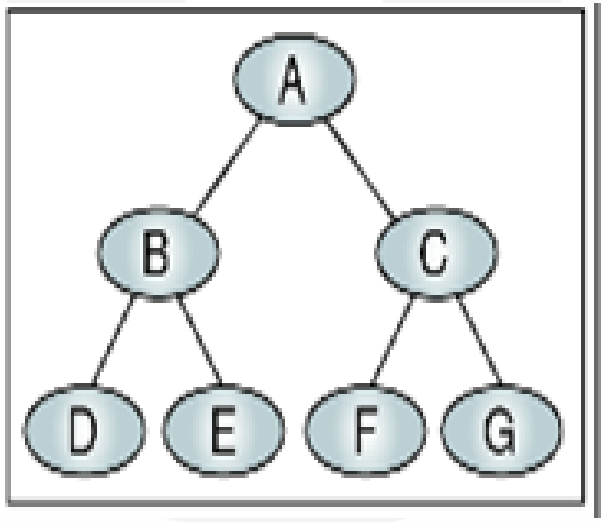
Some Properties of Binary Trees

- A binary tree with N nodes has exactly N-1 edges.
- The number of leaf nodes is equal to the number of nodes with two children + 1.

$$N_{\max} = 2^{H} - 1$$

Total edges : 6

Leaf nodes : 4



Parul®
University

DIGITAL LEARNING CONTENT

Some Properties of Binary Trees

•The balance factor of a binary tree is the difference in height between its left and right sub trees:

$$B = H_L - H_R$$

Some Properties of Binary Trees

Balance of the tree

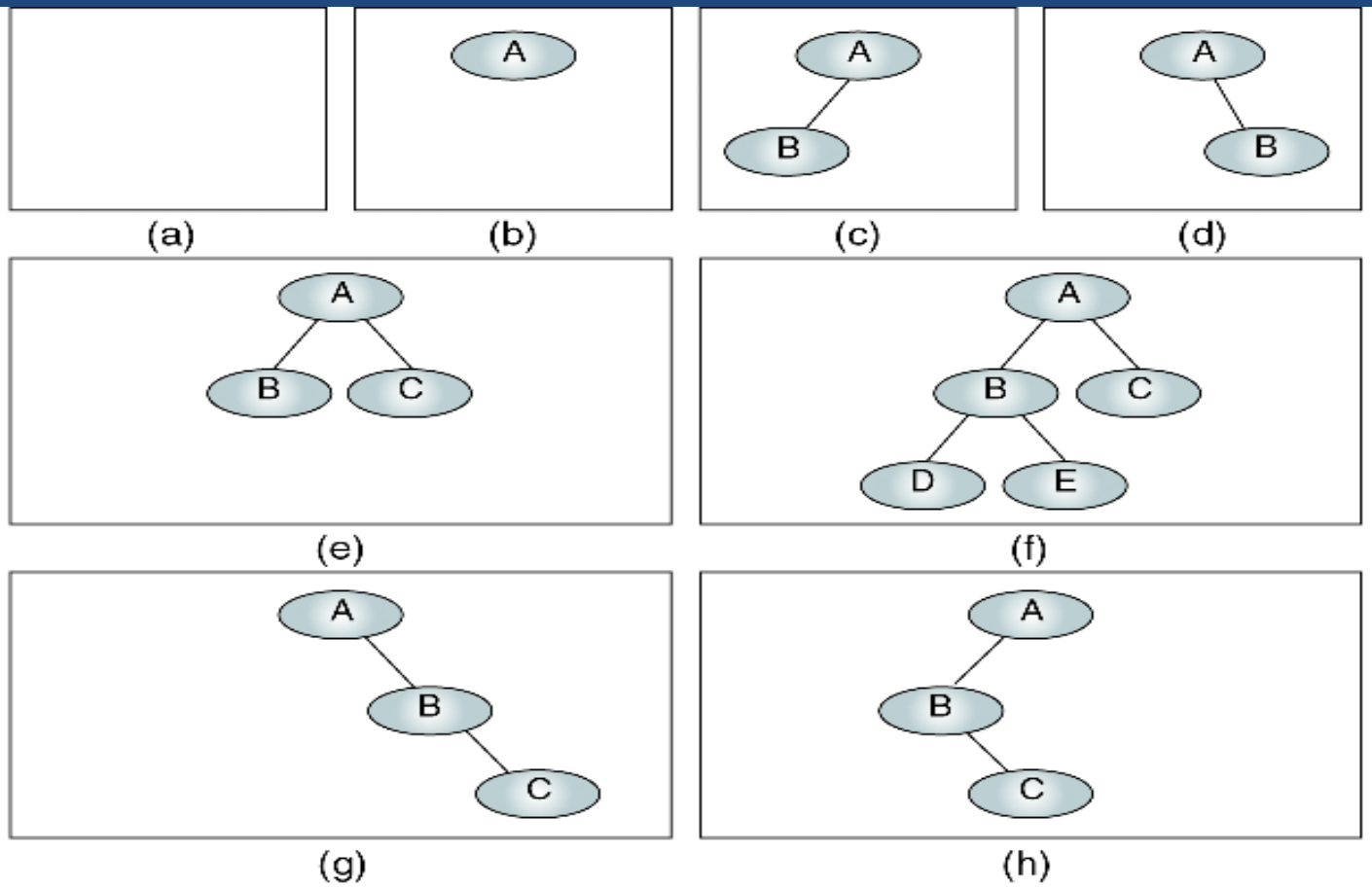
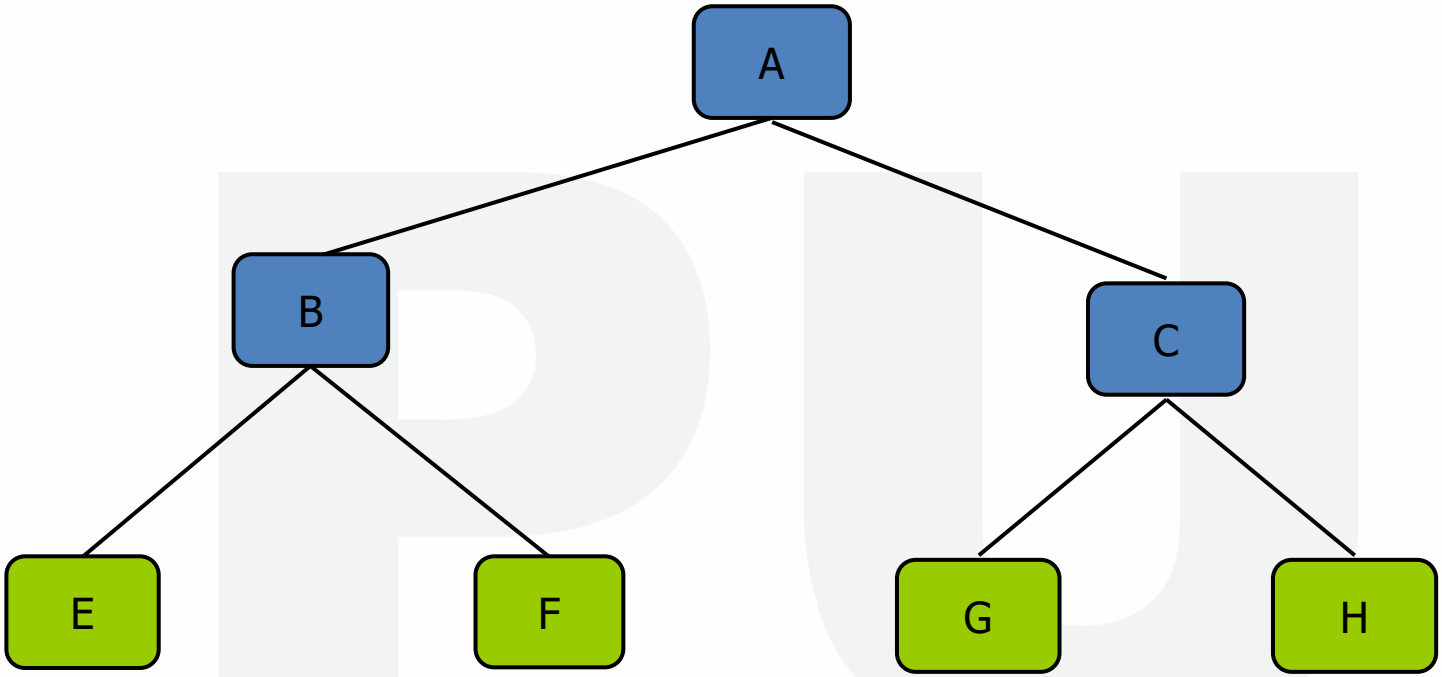


FIGURE 6-6 Collection of Binary Trees

Complete binary trees

- A **complete binary tree** is tree in which all the leaves are at same level and any non-leaf node have exactly two children.
- The total number of nodes in complete binary trees are $2^h - 1$.
- Total number of leaf nodes in complete binary trees are 2^{h-1} .
- Non leaf node = $2^{h-1} - 1$.
- At each level there are exactly 2^L number of nodes.

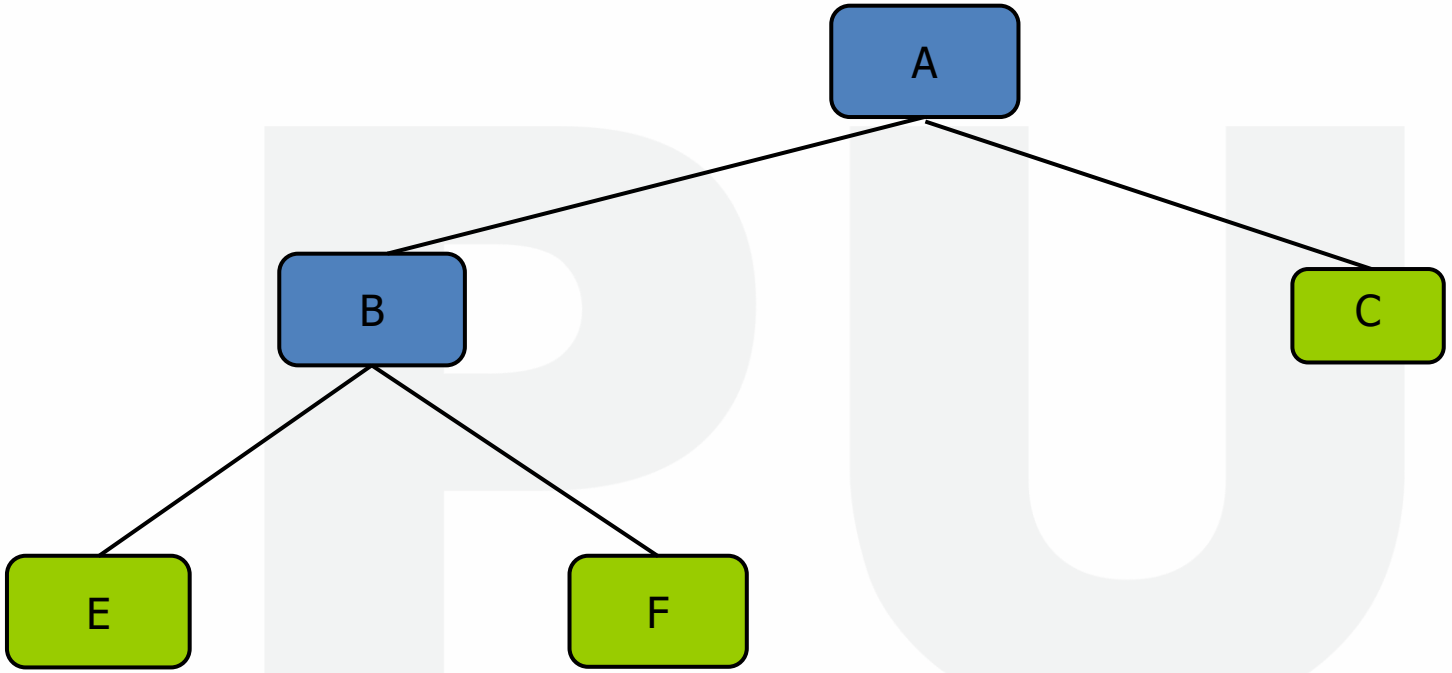
Complete binary trees



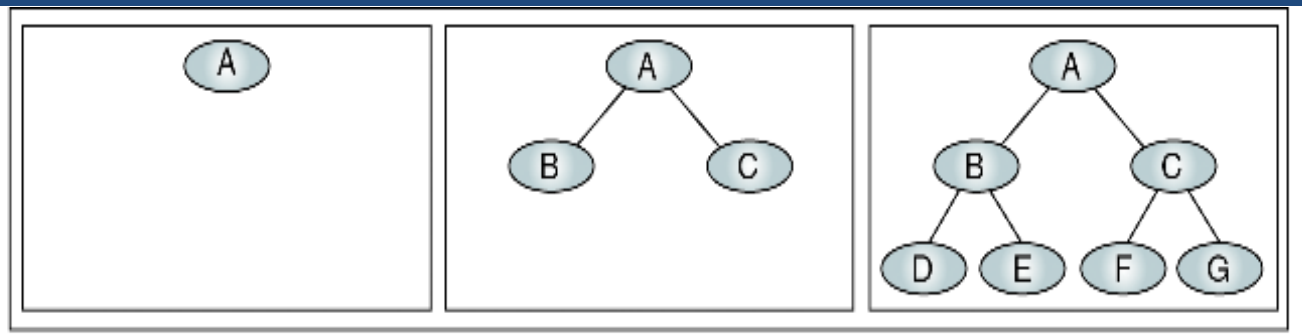
strictly binary trees

- When every non-leaf node in **binary tree** is filled with left and right subtrees, then a tree is called strictly binary tree.
- In other word a node have either two child node or zero child node at all.
- A strictly binary tree with N leaves always have **2N-1** nodes.

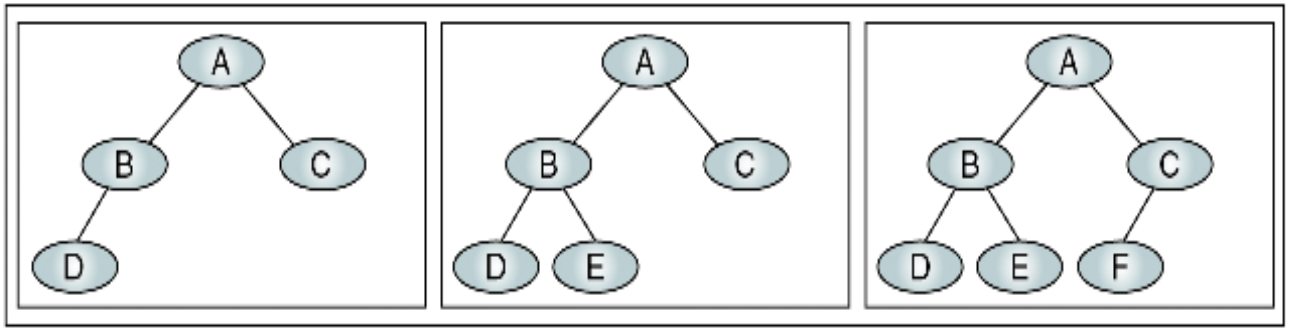
strictly binary trees



strictly binary trees



(a) Complete trees (at levels 0, 1, and 2)

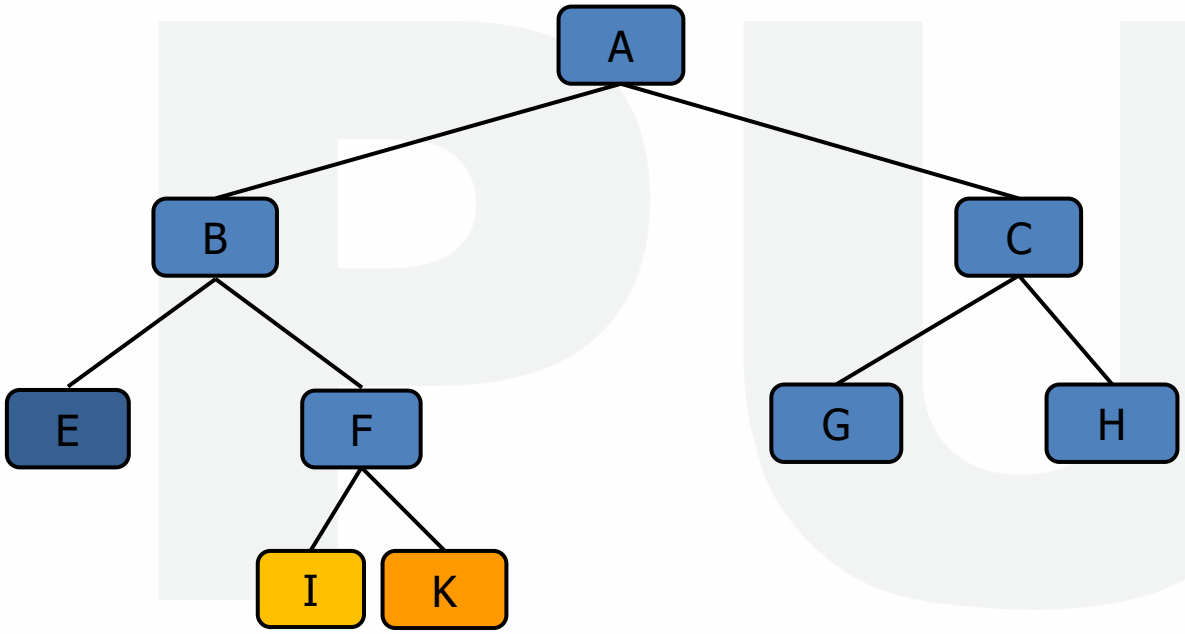


(b) Nearly complete trees (at level 2)

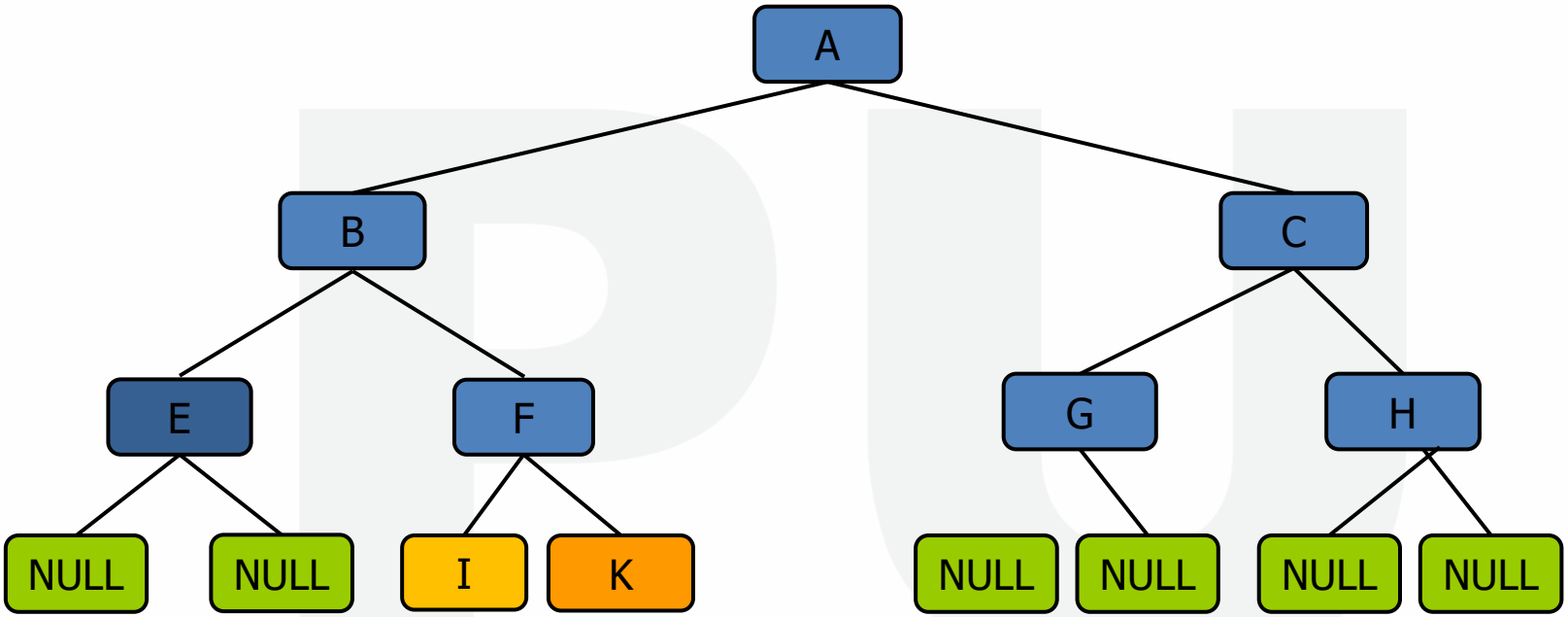
FIGURE 6-7 Complete and Nearly Complete Trees

Representation of a trees

- Array representation
- Linked List representation

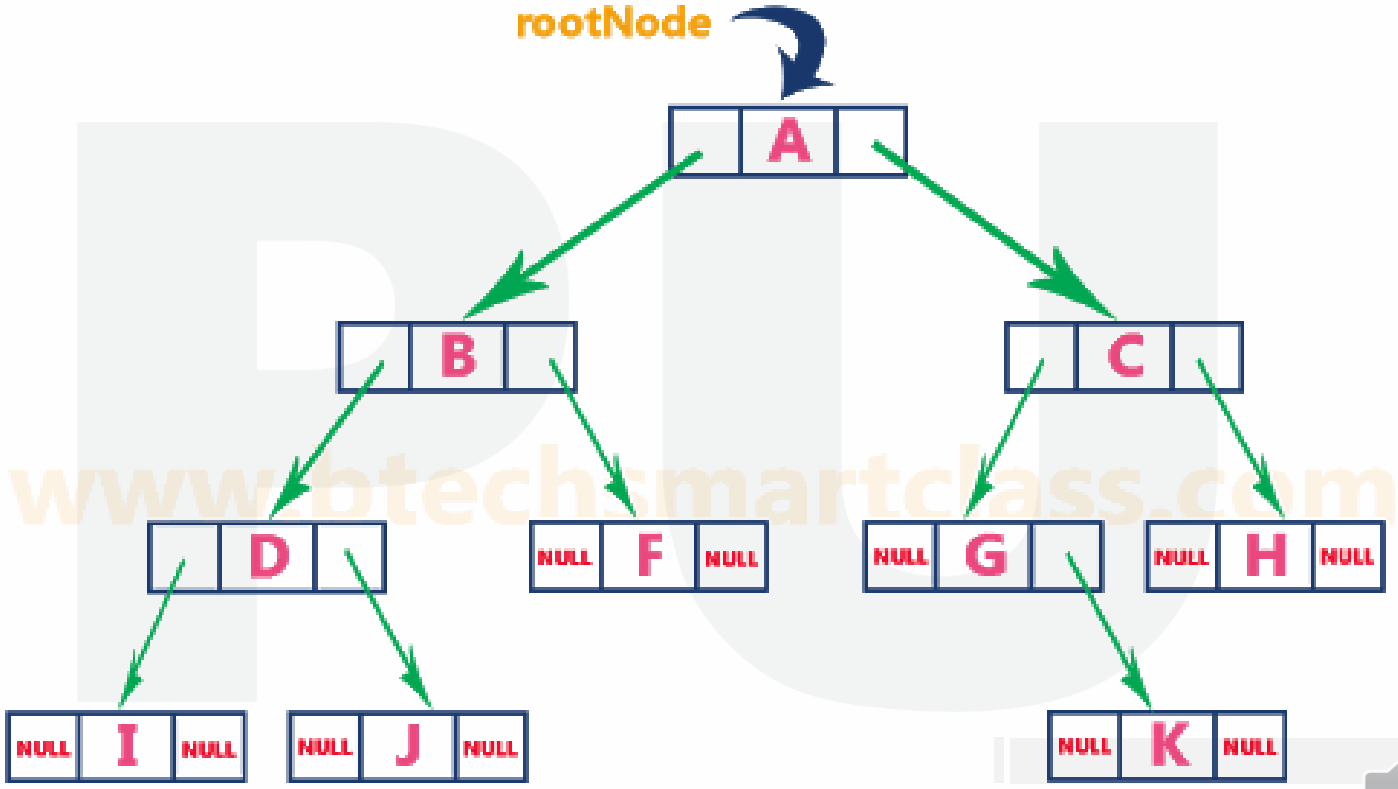


Array Representation of a trees



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	C	E	F	G	H			I	K				

Linked Representation of a trees



Parul®
University

DIGITAL LEARNING CONTENT

Binary trees Transversal

•A **binary tree traversal** requires that each node of the tree be processed once and only once in a predetermined sequence.

Trees Transversal

1. In-order traversal

   Left sub tree -------- root--------right sub tree

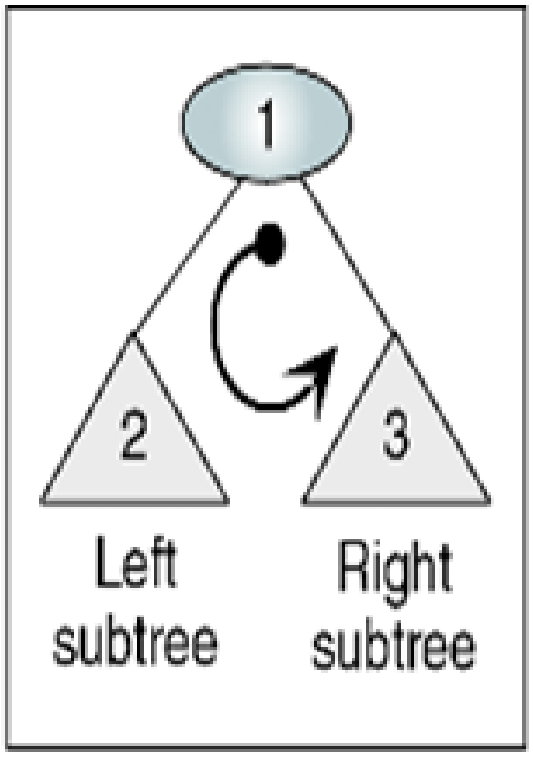
2. Pre-order traversal

   root-------- Left sub tree -------- right sub tree

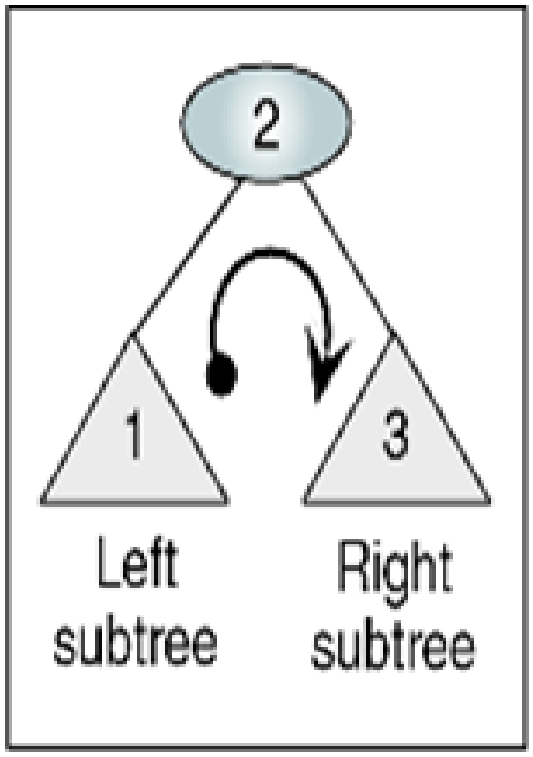
3. Post order traversal

   Left sub tree-------right sub tree -------- root

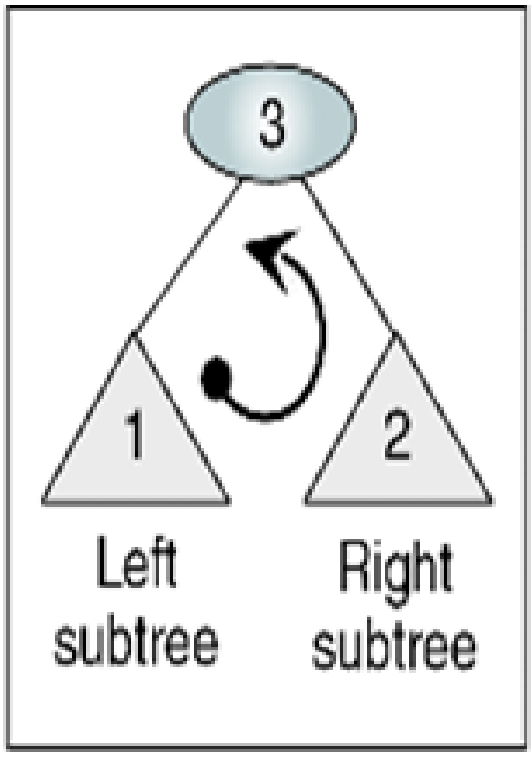
Trees Transversal



(a) Preorder traversal

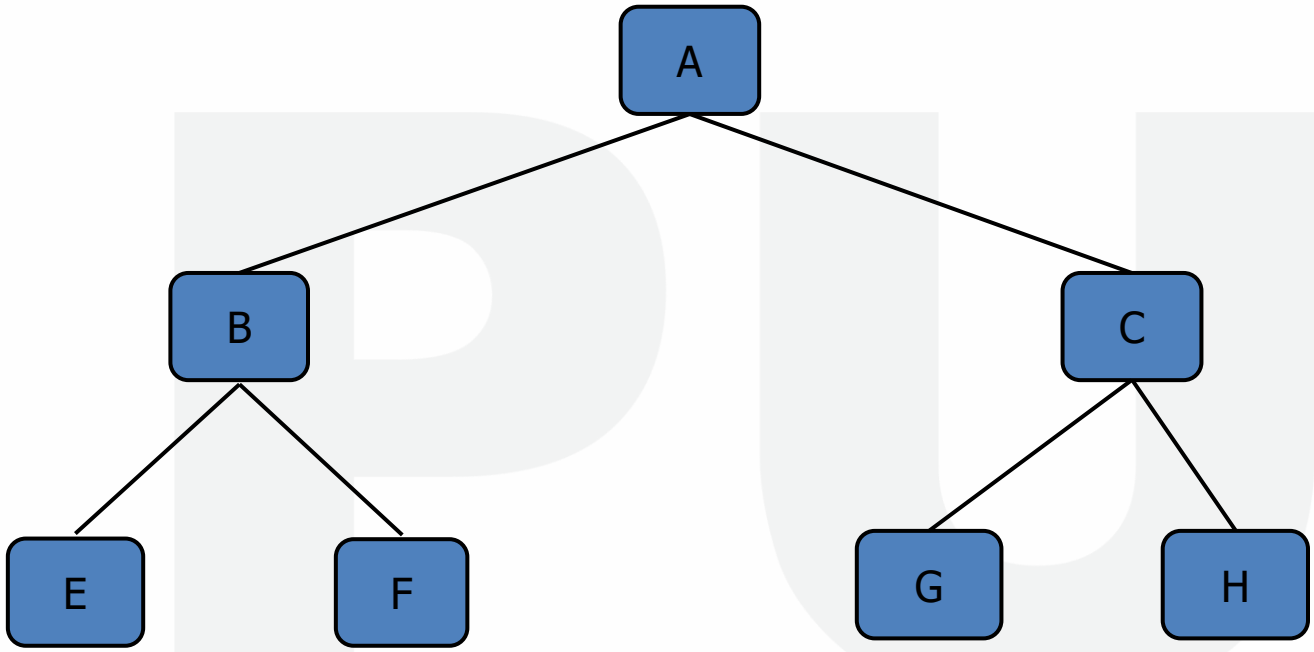


(b) Inorder traversal



(c) Postorder traversal

Trees Transversal



Parul®
University

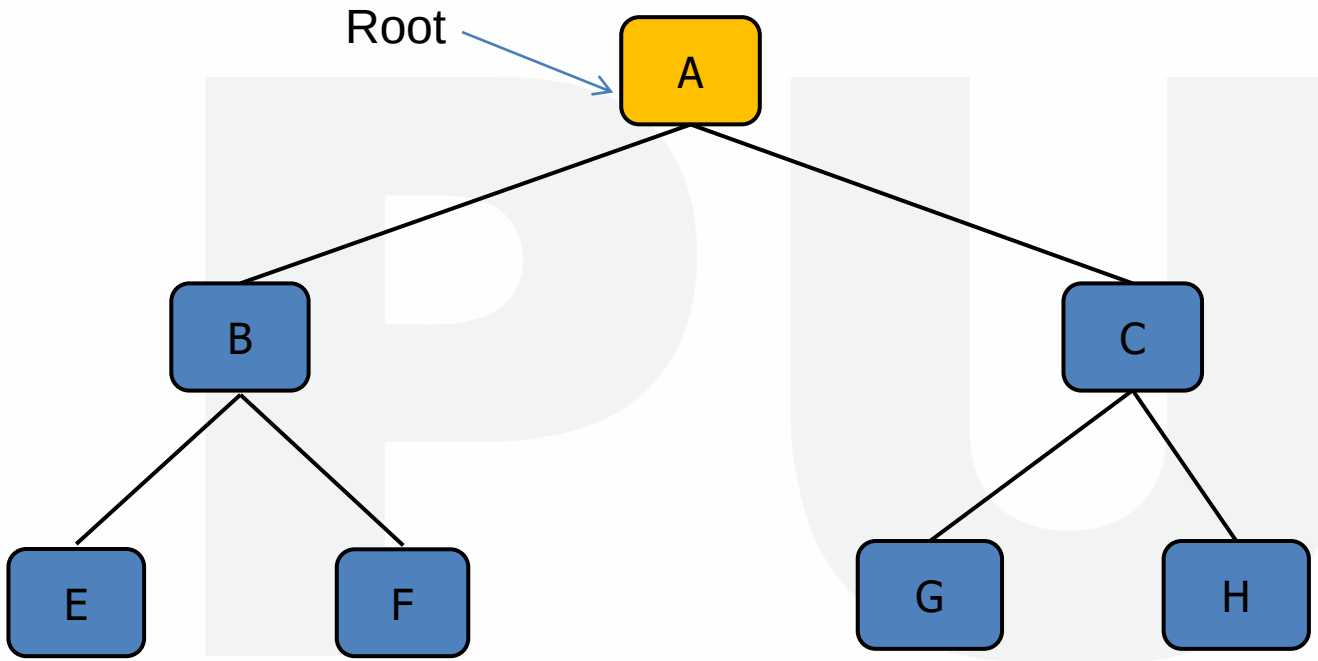
DIGITAL LEARNING CONTENT

Trees Transversal

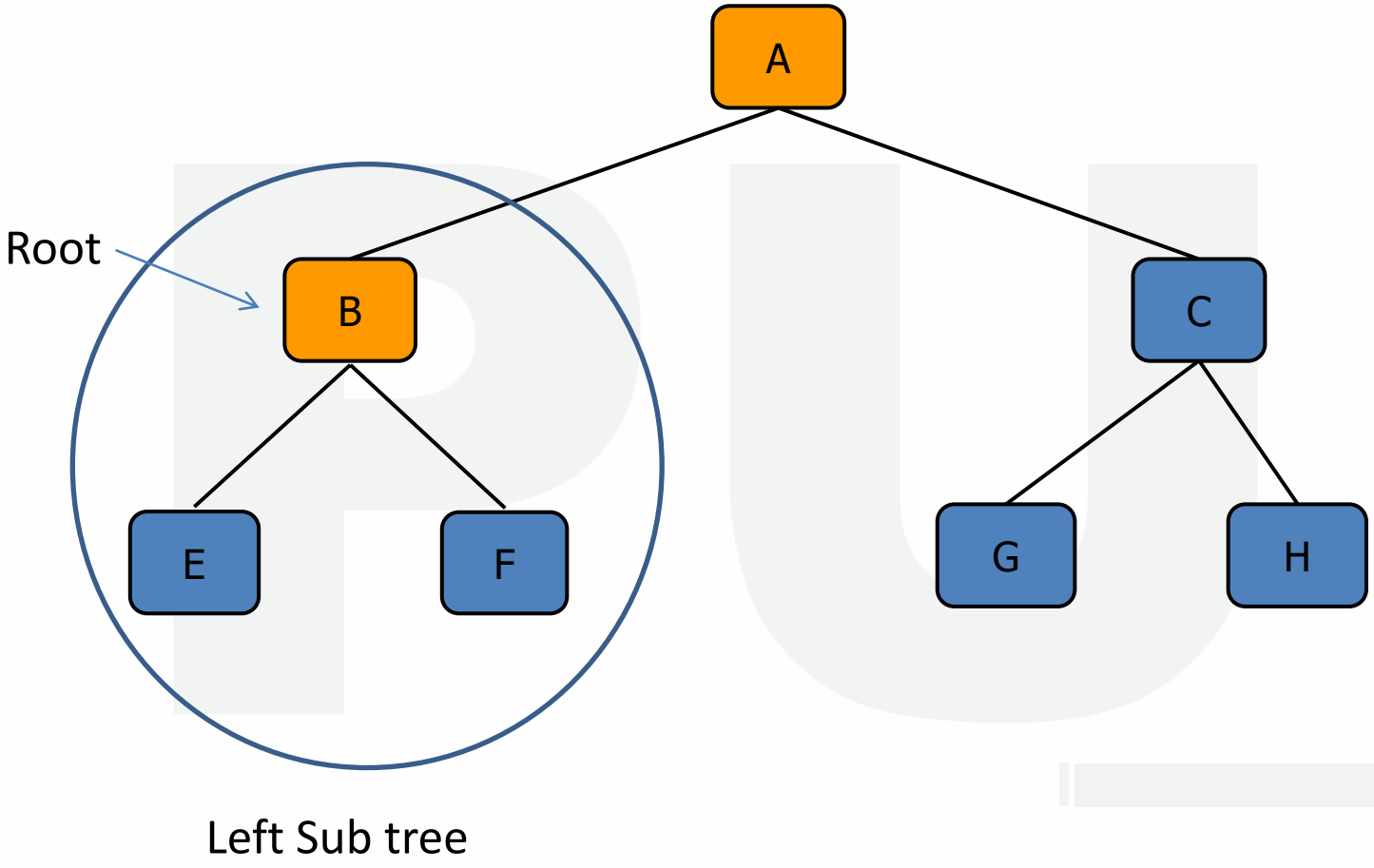
ALGORITHM 6-2 Preorder Traversal of a Binary Tree

```
Algorithm preOrder (root)
Traverse a binary tree in node-left-right sequence.
   Pre  root is the entry node of a tree or subtree
   Post each node has been processed in order
1  if (root is not null)
   1   process (root)
   2   preOrder (leftSubtree)
   3   preOrder (rightSubtree)
2  end if
end preOrder
```

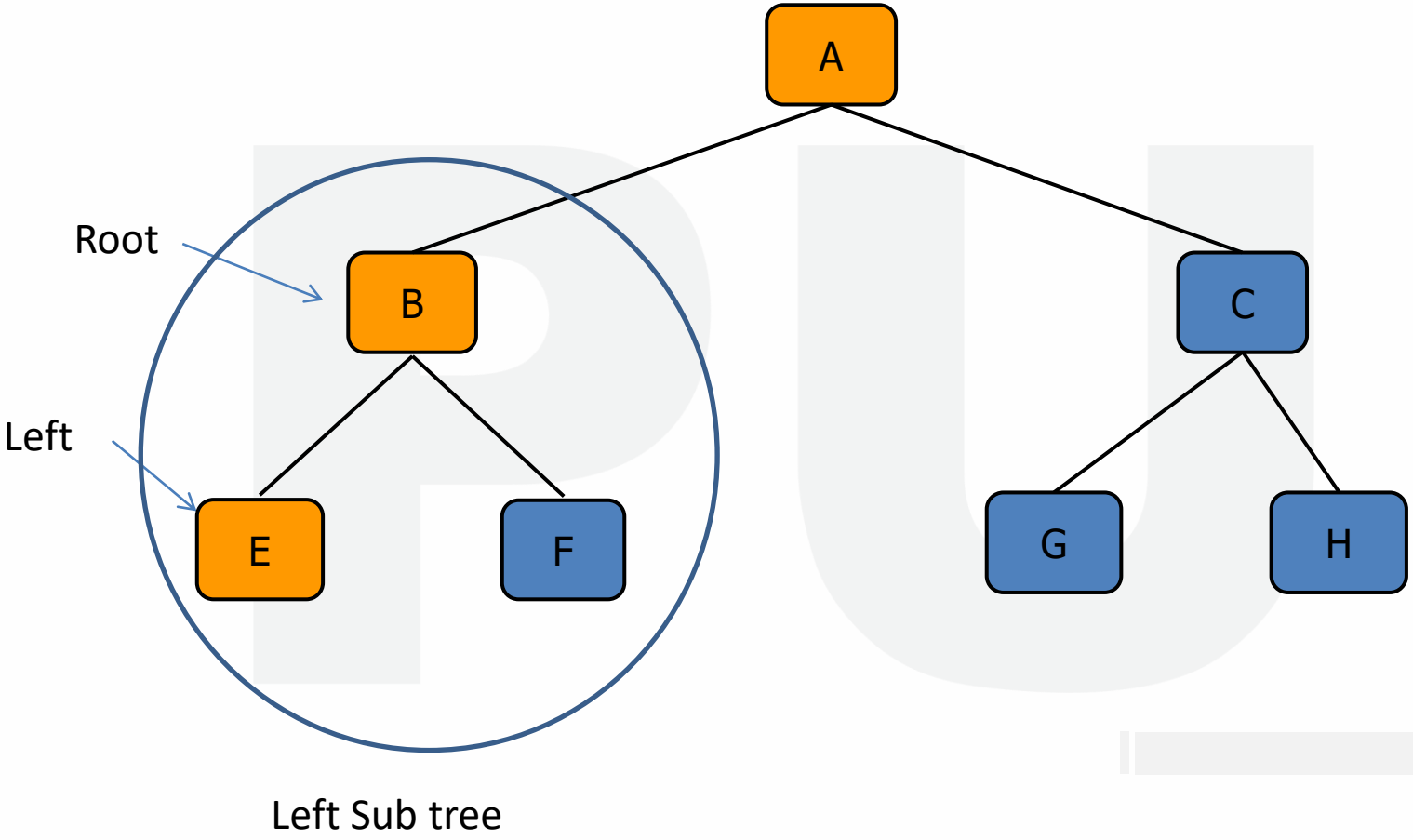
Trees Transversal



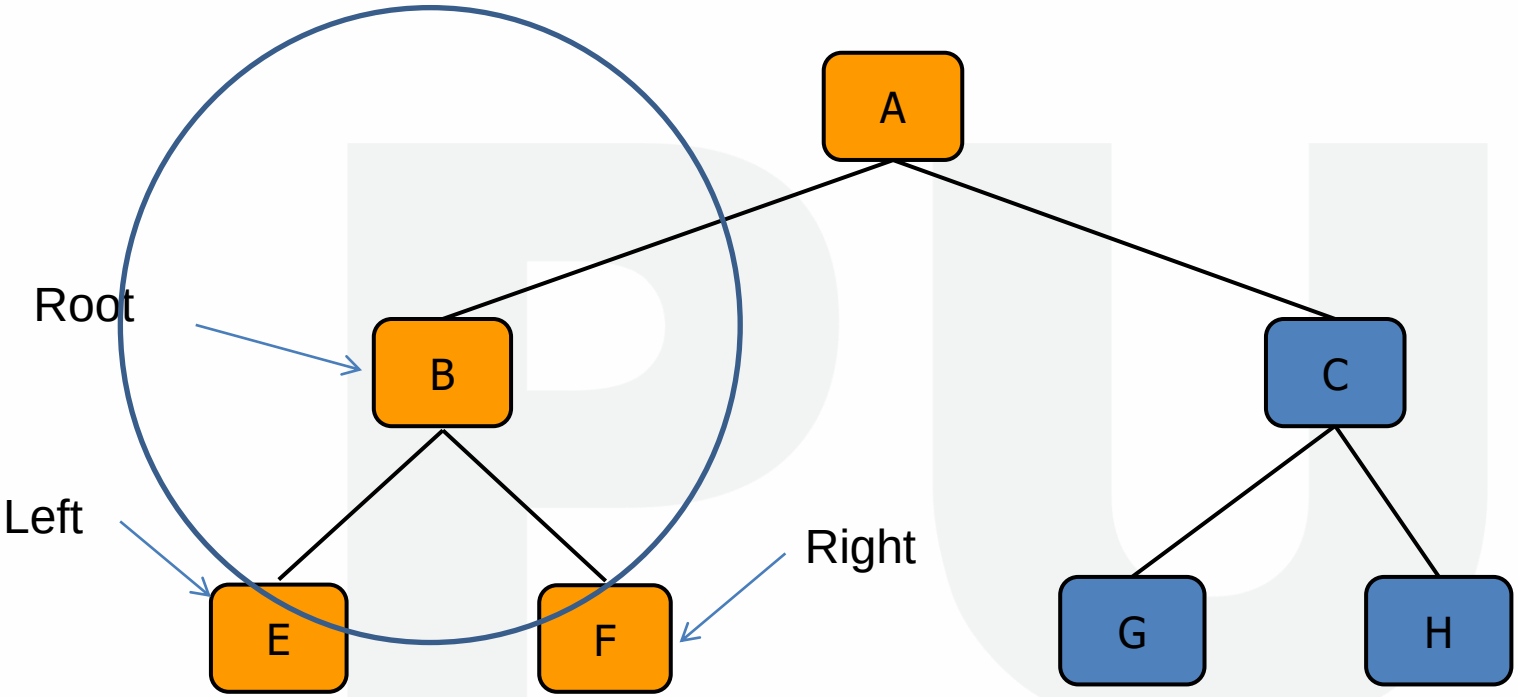
Trees Transversal



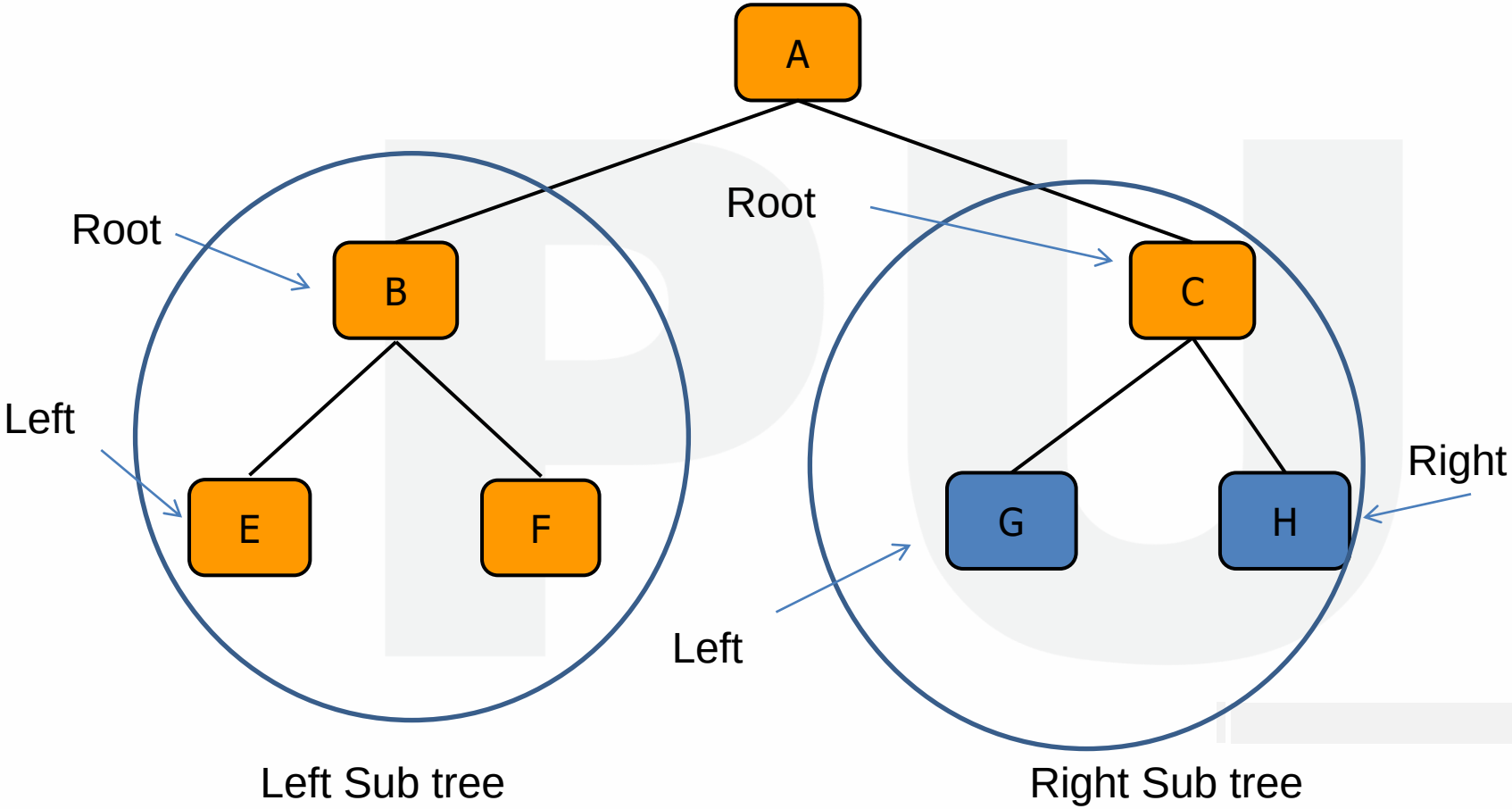
Trees Transversal



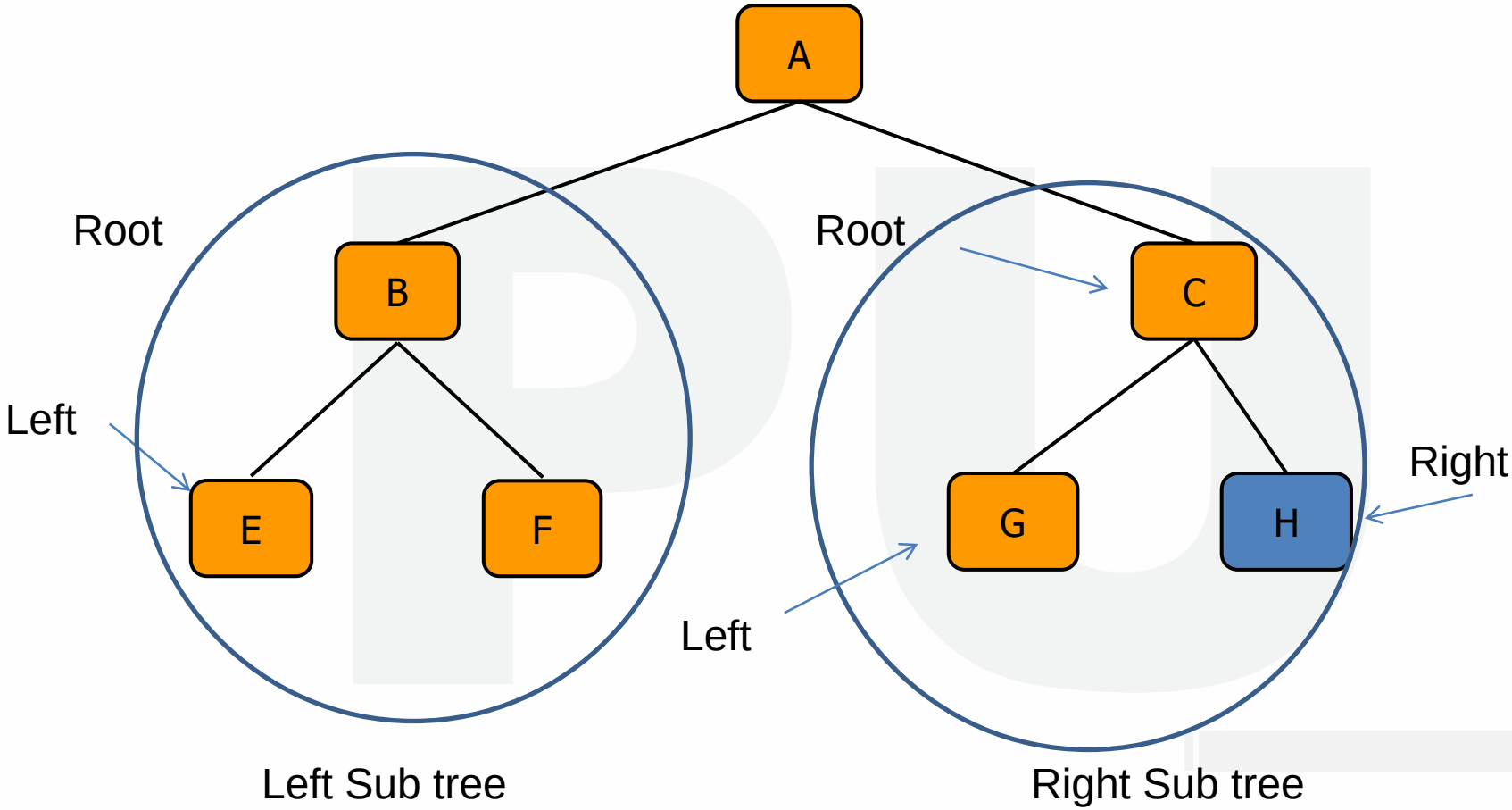
Trees Transversal



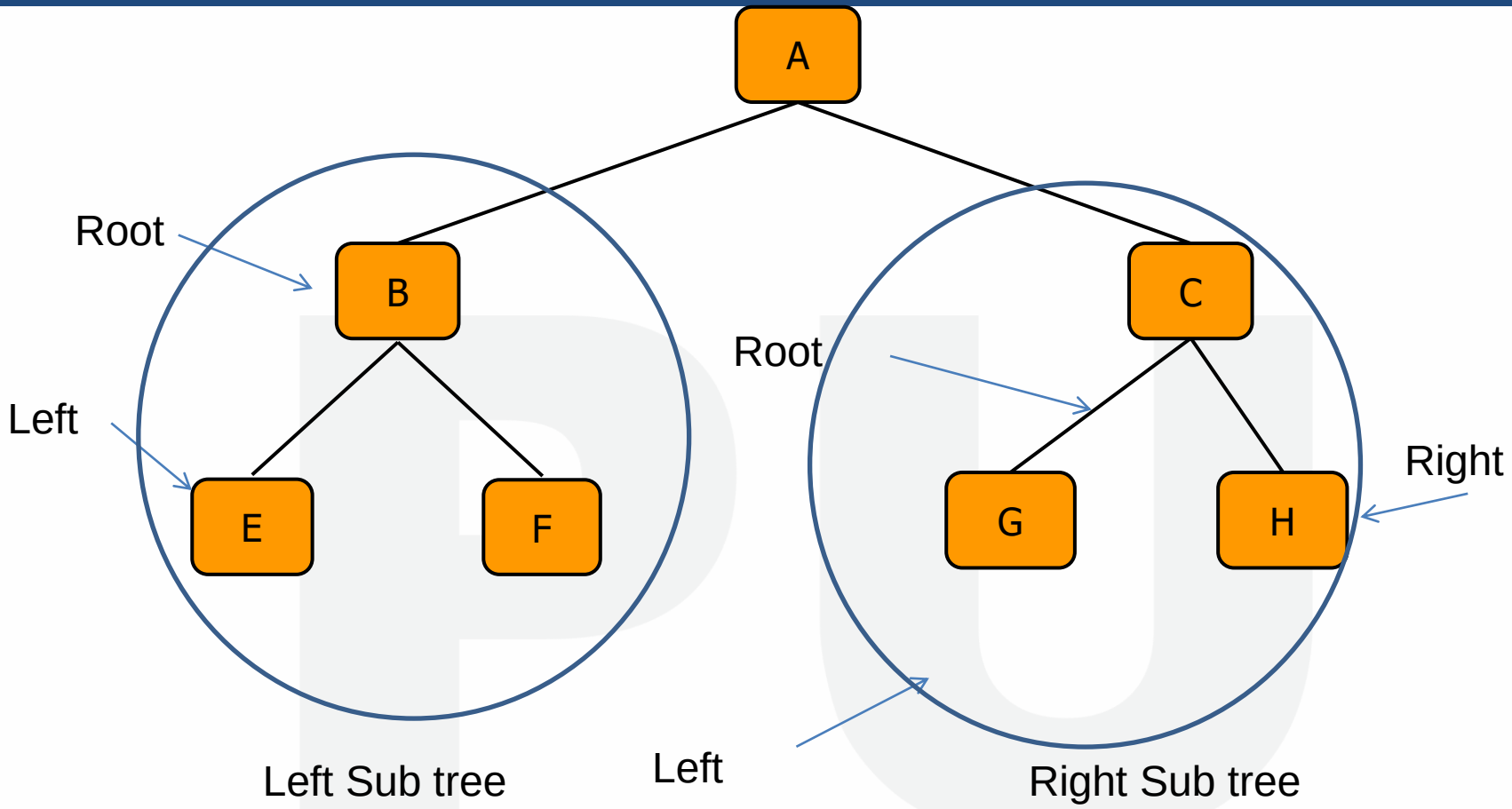
Trees Transversal



Trees Transversal

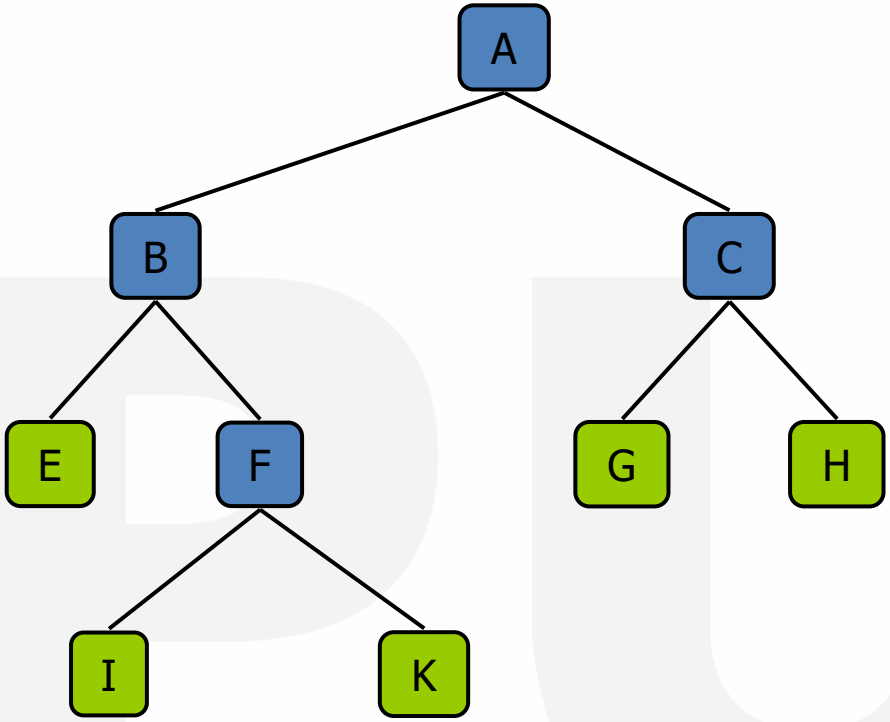


Trees Transversal



Preorder : A B E F C G H

Find the preorder traversal of following tree



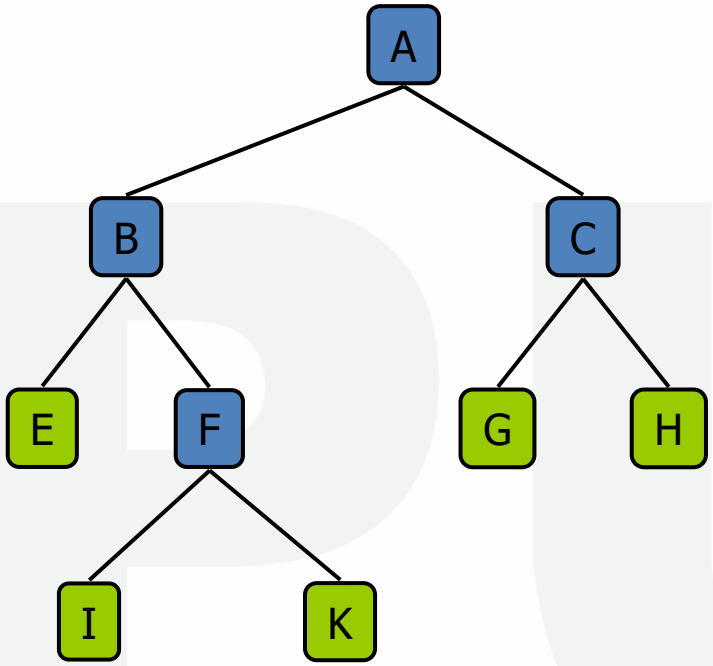
A,B,E,F,I,K,C,G,H

Inorder traversal

ALGORITHM 6-3 Inorder Traversal of a Binary Tree

```
Algorithm inOrder (root)
Traverse a binary tree in left-node-right sequence.
   Pre  root is the entry node of a tree or subtree
   Post each node has been processed in order
1  if (root is not null)
   1   inOrder (leftSubTree)
   2   process (root)
   3   inOrder (rightSubTree)
2  end if
end inOrder
```

Find the Inorder traversal of the following

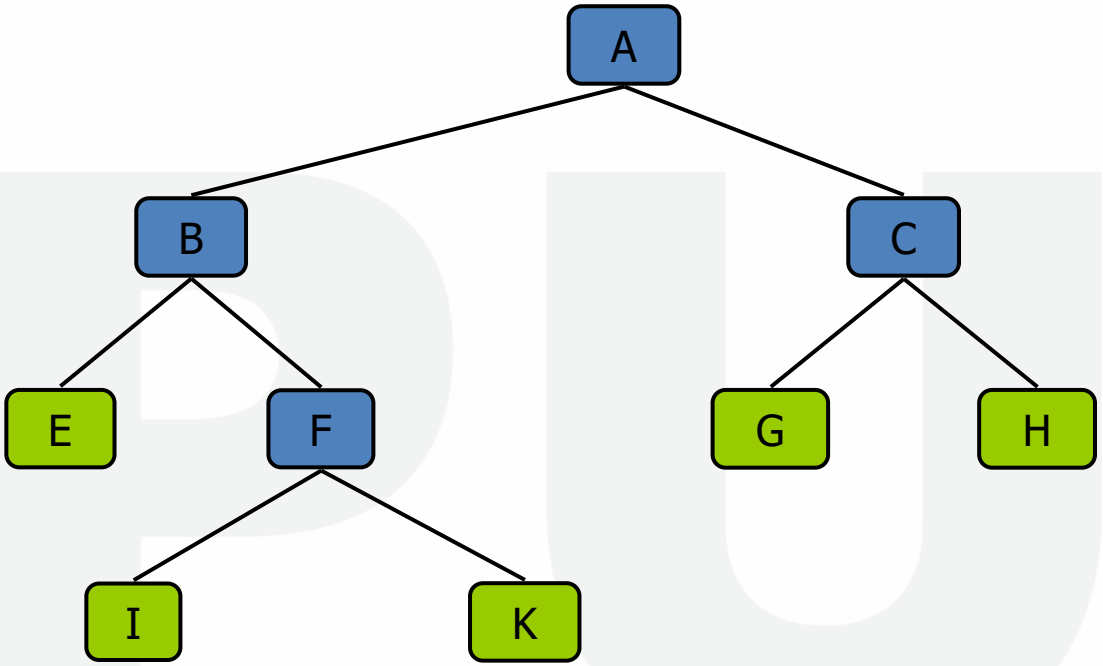


Postorder traversal Algorithm

ALGORITHM 6-4 Postorder Traversal of a Binary Tree

```
Algorithm postOrder (root)
Traverse a binary tree in left-right-node sequence.
   Pre  root is the entry node of a tree or subtree
   Post each node has been processed in order
1  if (root is not null)
   1   postOrder (left subtree)
   2   postOrder (right subtree)
   3   process (root)
2  end if
end postOrder
```

Find the Postorder traversal of the following tree



a x (b + c) + d

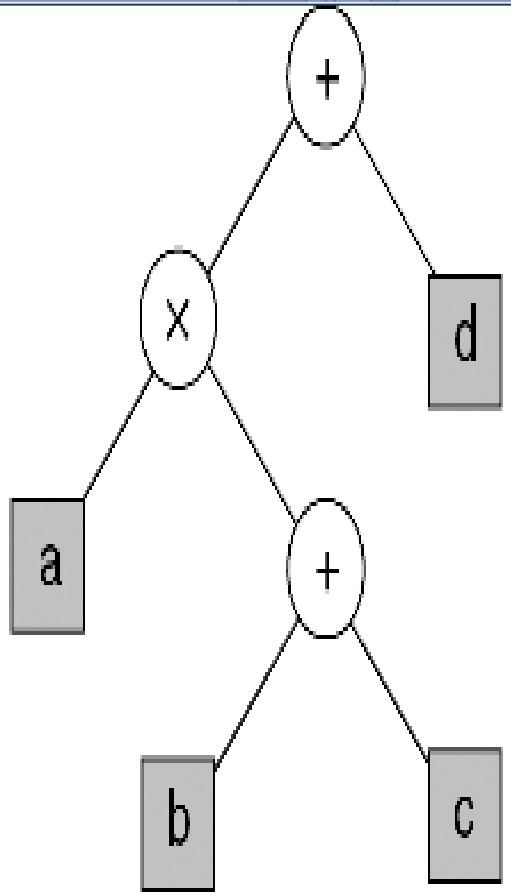


FIGURE 6-15 Infix Expression and Its Expression Tree

Reconstruct Binary tree

We can reconstruct binary from following two traversals

1) In order traversal
2) Pre-order or post order traversal

Lets reconstruct binary from following two traversals

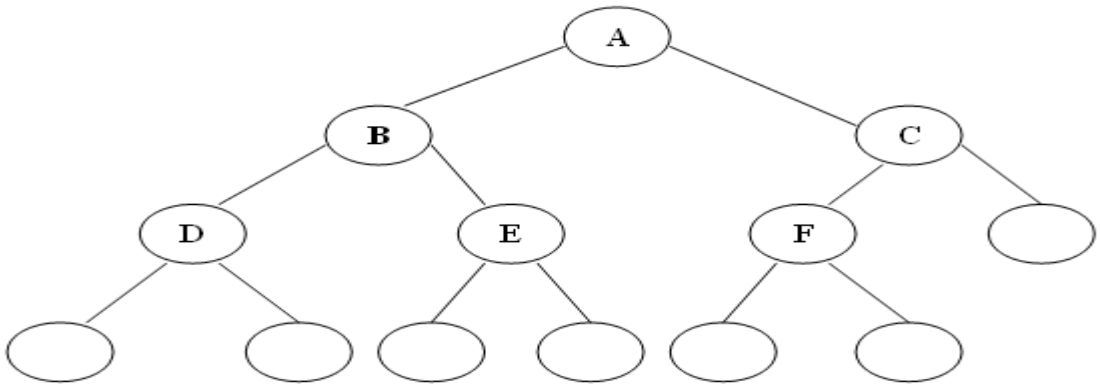
1) In order traversal: XBGDHAEICJFK
2) Pre-order traversal: ABXDGHCEIFJK

Lets reconstruct binary from following two traversals

1) In order traversal: bgdaecf
2) Pre-order traversal: gdbefca

Threaded Binary tree

- In a linked representation of a binary tree, the number of null links (null pointers) are actually more than non-null pointers.
- These null links can be replace with pointers called threads.
- Consider the following binary tree:



A Binary tree with the null pointers

Threaded Binary tree

- A left null link of a node is replaced with the address of its inorder predecessor.
- And right null link of a node is replace with the address of its inorder successor.
- In previous example there are 7 null pointers & 5 Actual.
- We can generalize it that for any binary tree with n nodes there will be (n+1) null pointers and 2n total pointers.
- The objective here to make effective use of these null pointers.

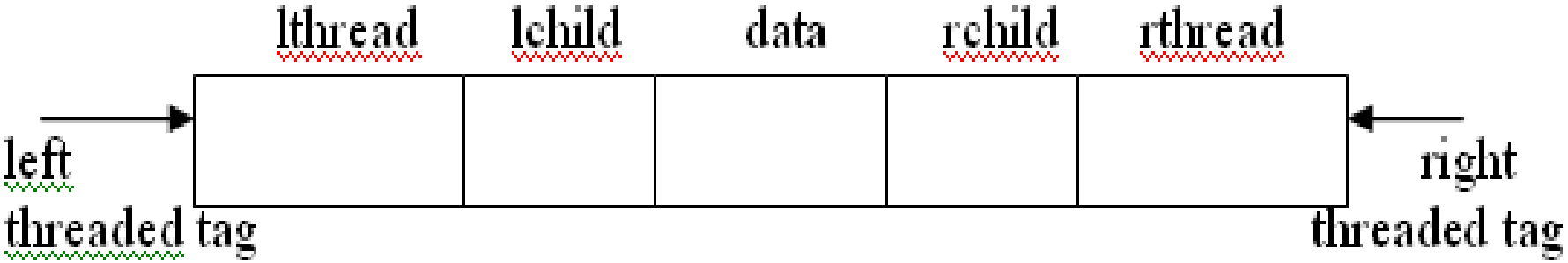
Threaded Binary tree

- And binary tree with such pointers are called threaded binary tree.

- In the memory representation of a threaded binary tree, it is necessary to distinguish between a normal pointer and a thread.

Threaded Binary tree

•Therefore we have an alternate node representation for a threaded binary tree which contains five fields as show bellow



For any node p, in a threaded binary tree.

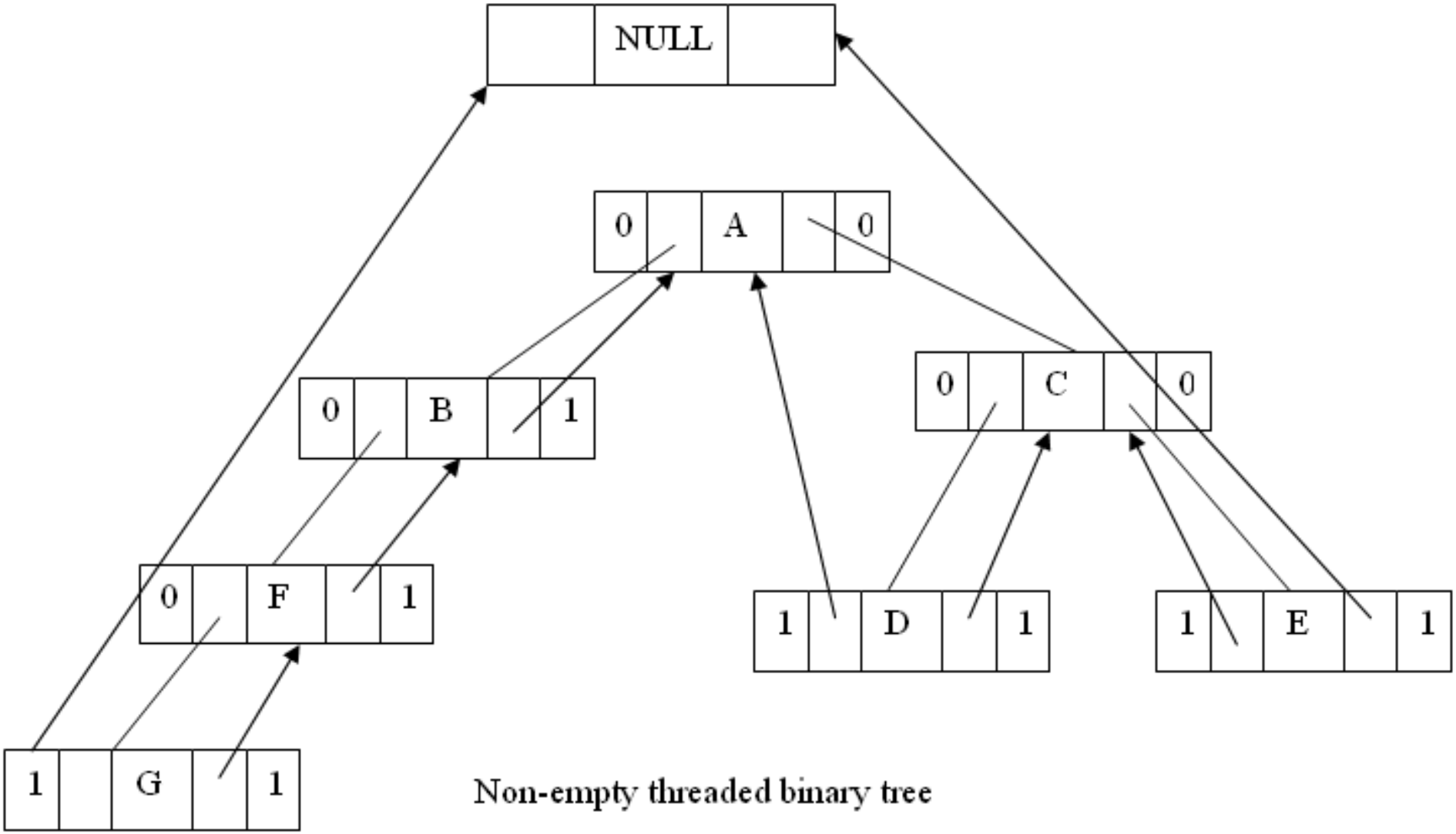
lthred(p)=1 indicates lchild (p) is a thread pointer

lthred(p)=0 indicates lchild (p) is a normal

rthred(p)=1 indicates rchild (p) is a thread

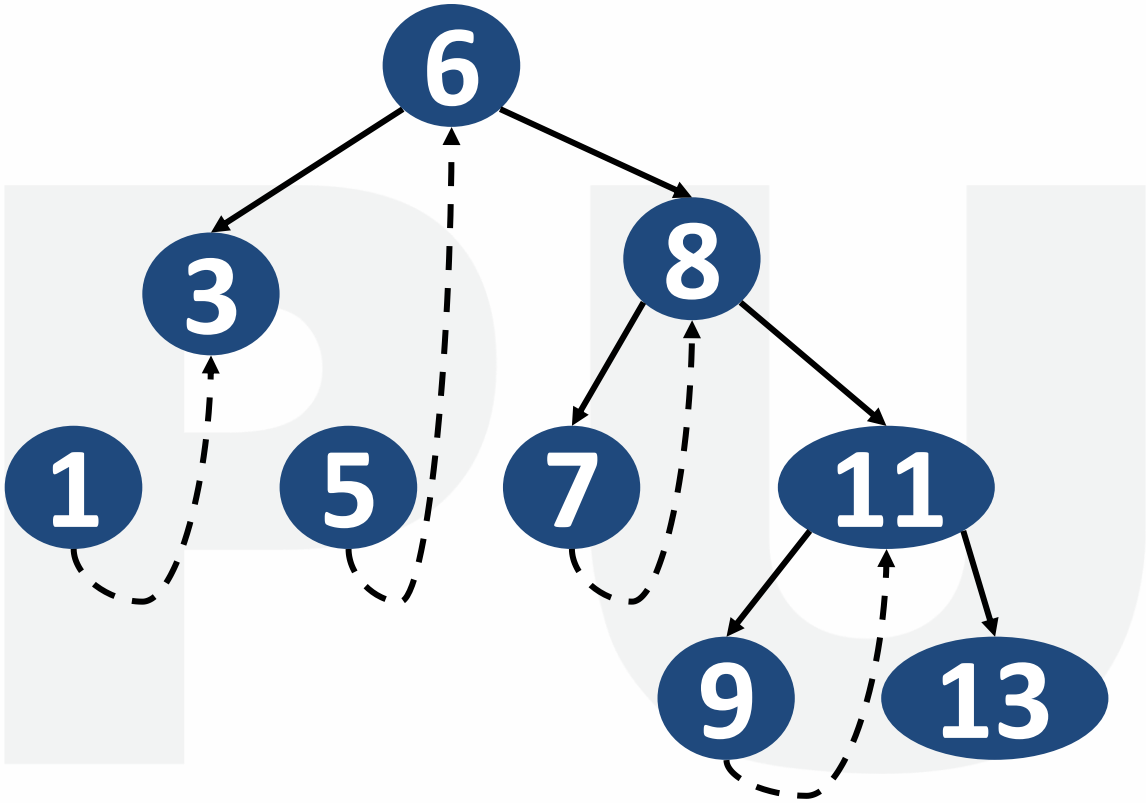
rthred(p)=0 indicates rchild (p) is a normal pointer

Threaded Binary tree



Non-empty threaded binary tree

Threaded tree Transversal



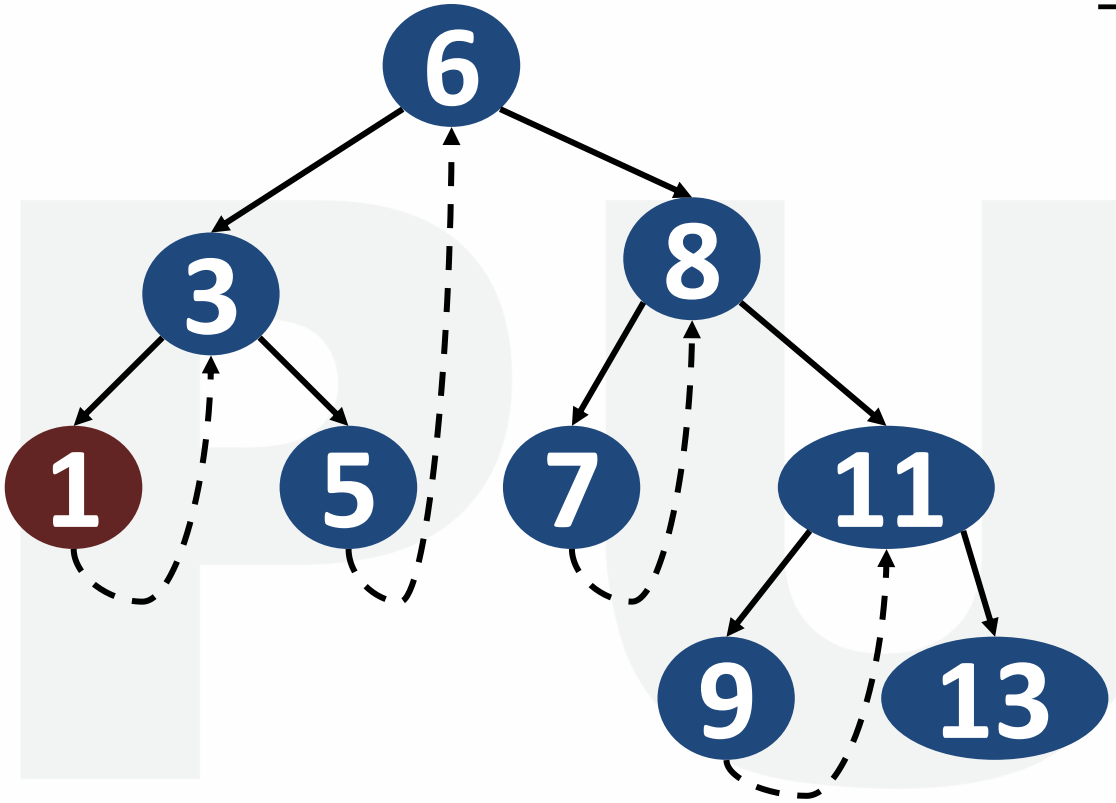
Threaded tree Transversal

- We start at the leftmost node in the tree, print it, and follow its right thread
- If we follow a thread to the right, we output the node and continue to its right
- If we follow a link to the right, we go to the leftmost node, print it, and continue

Threaded tree Transversal

Output

1



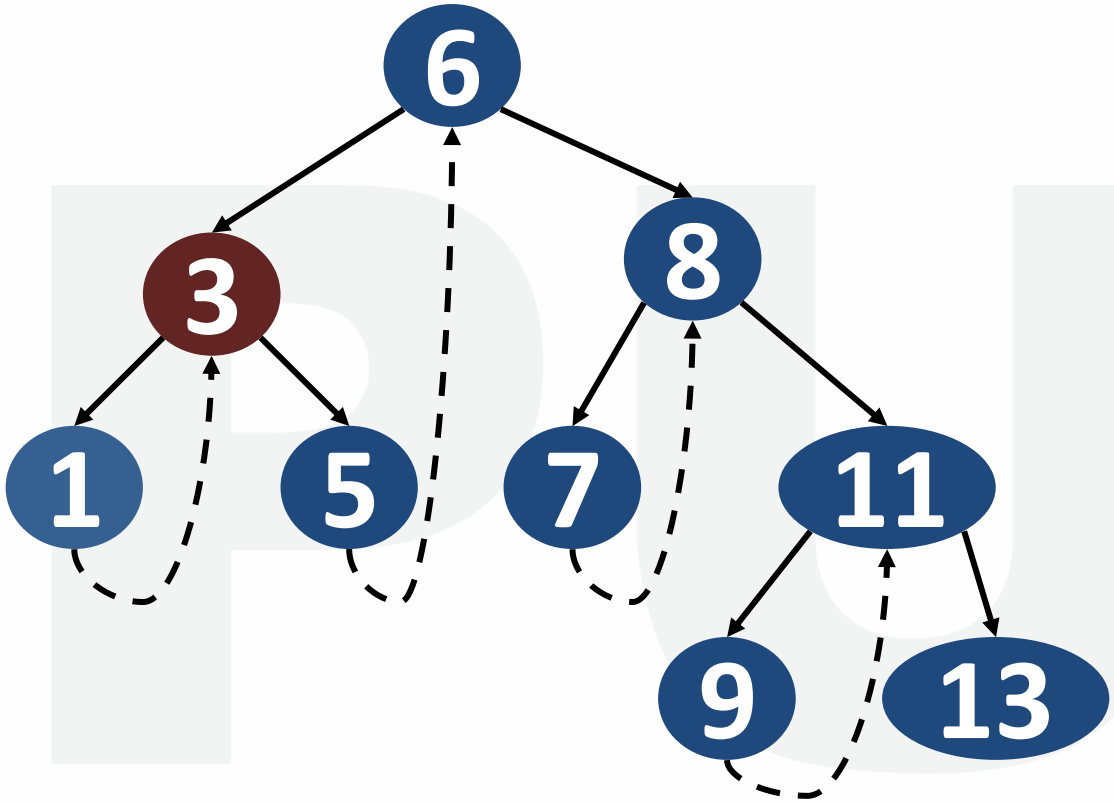
Start at leftmost node, print it

Threaded tree Transversal

Output

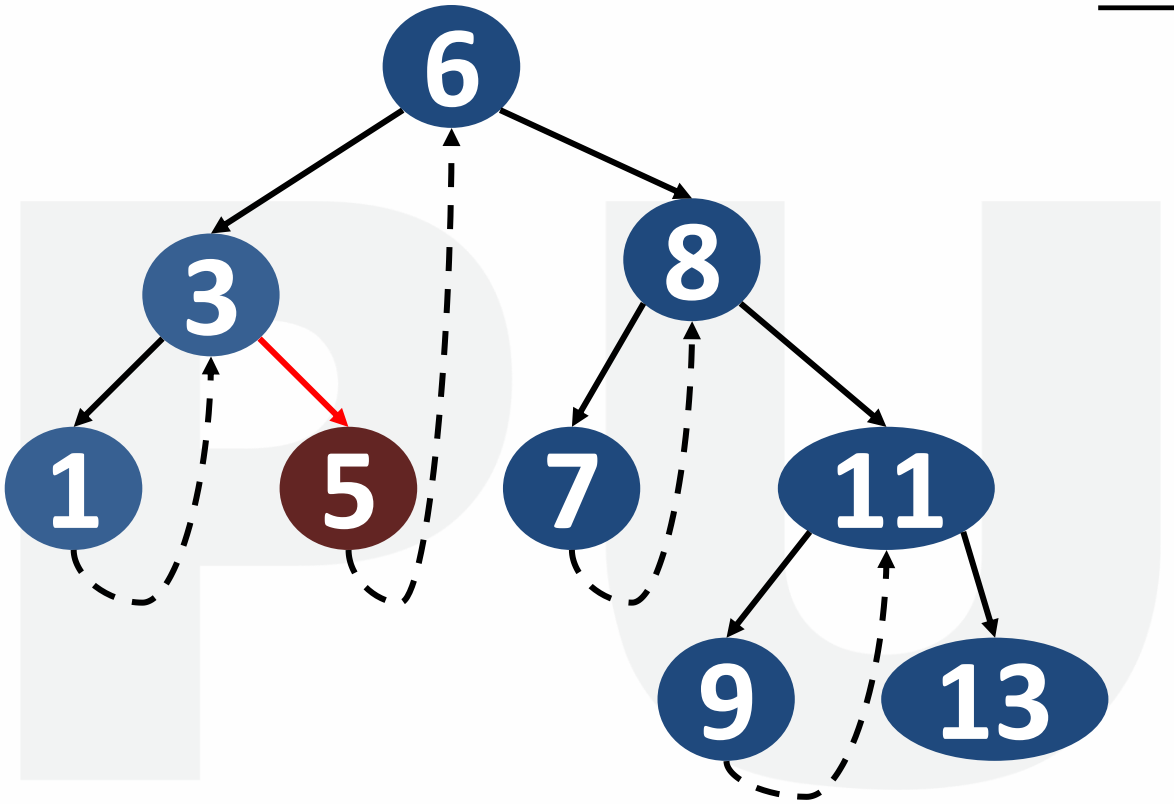
1

3



Follow thread to right, print node

Threaded tree Transversal



Output

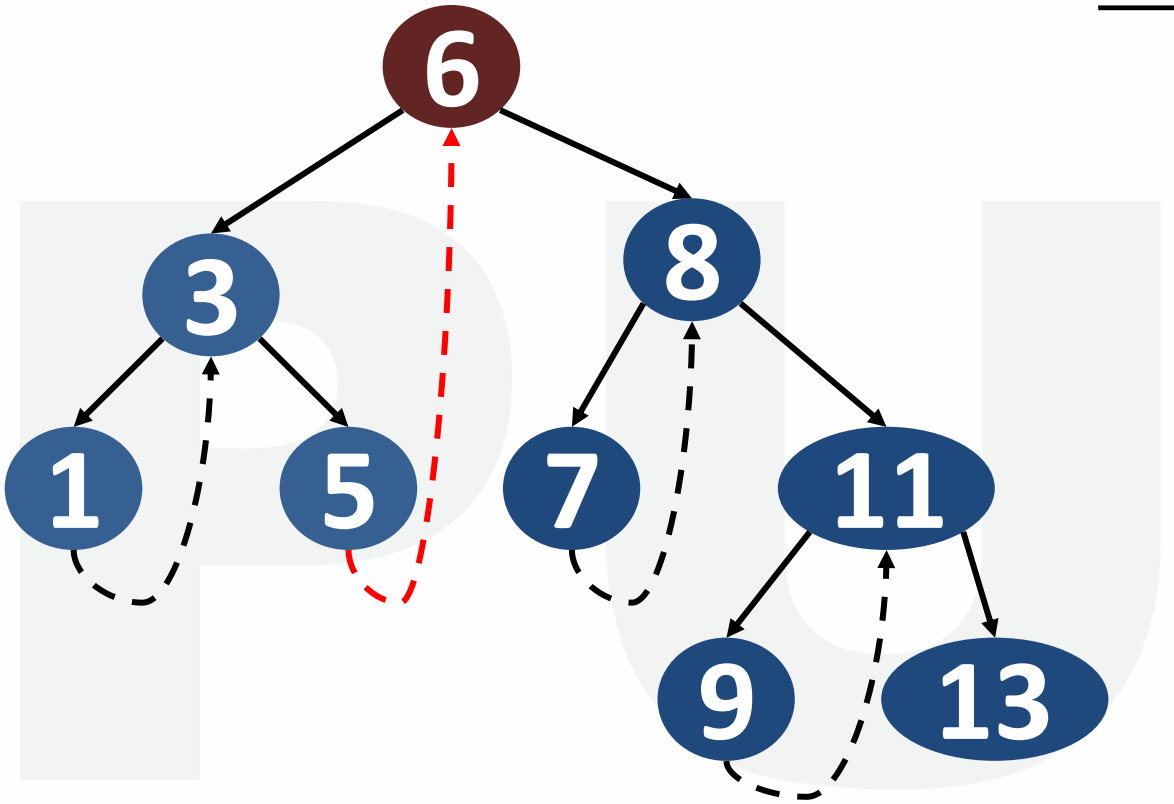
1

3

5

Follow link to right, go to leftmost node and print

Threaded tree Transversal



Output

1

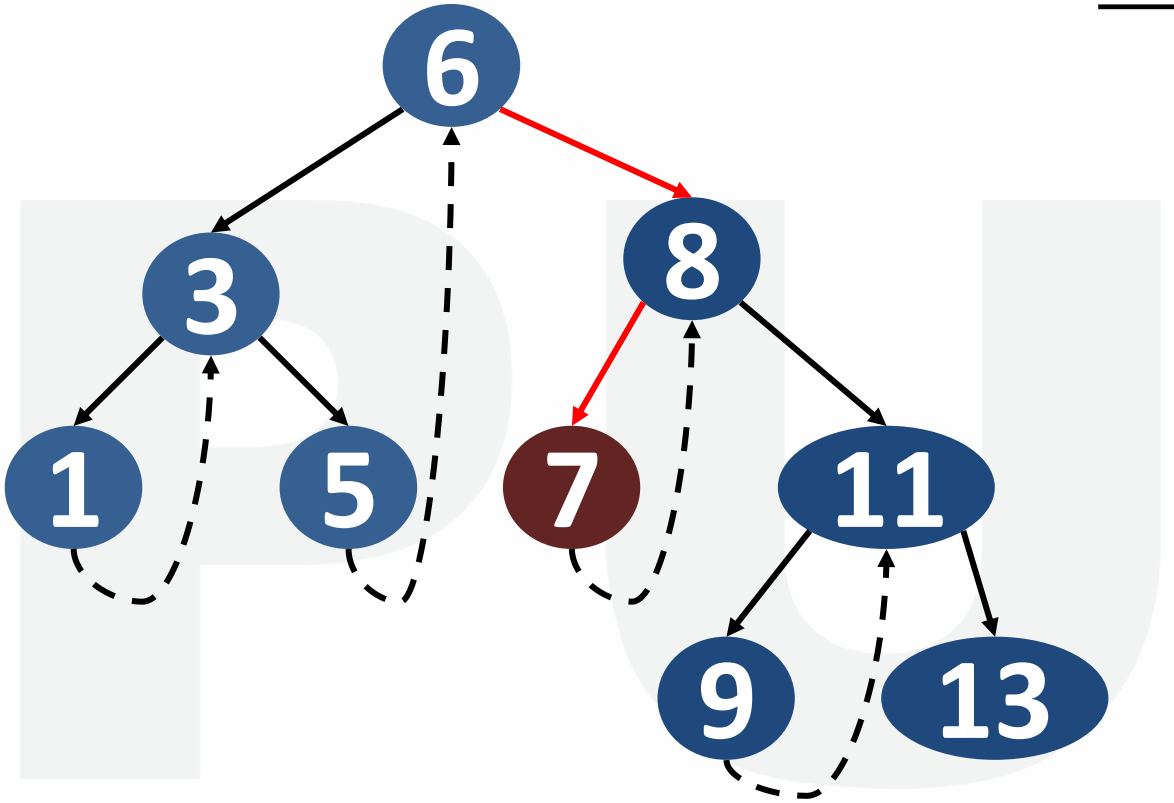
3

5

6

Follow thread to right, print node

Threaded tree Transversal

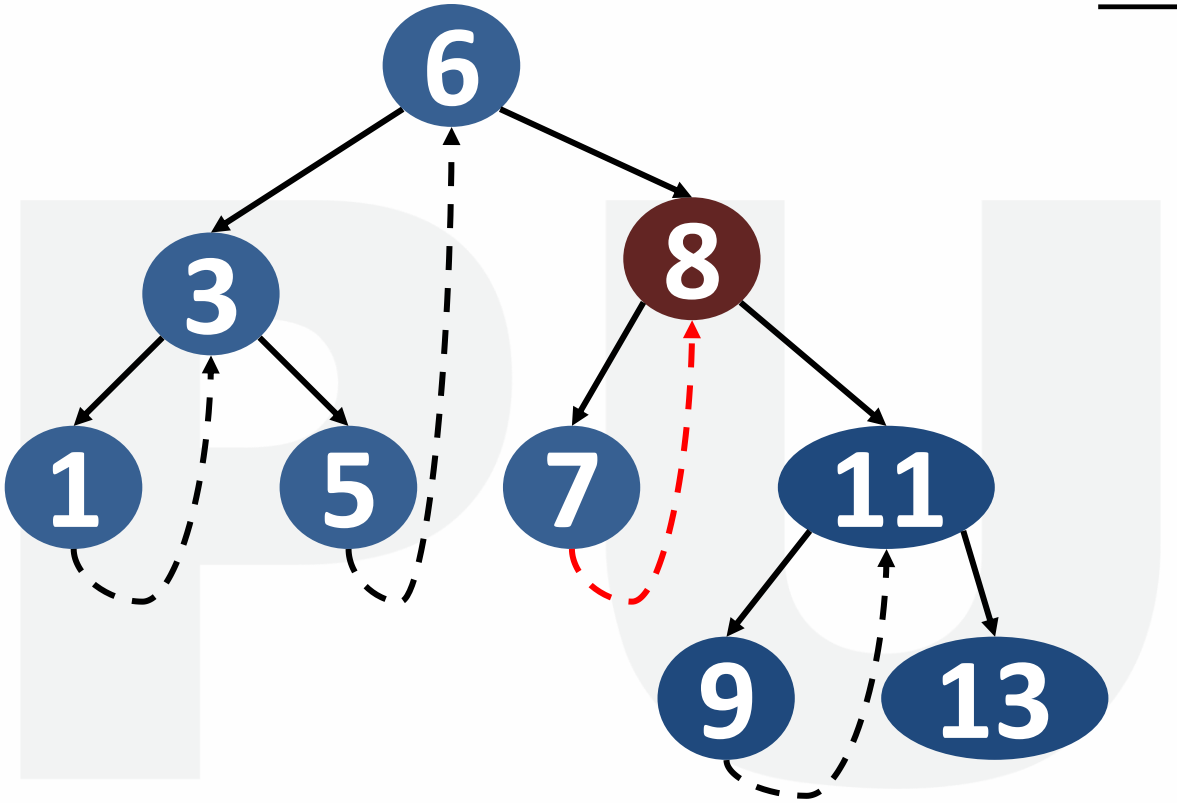


Output

1
3
5
6
7

Follow link to right, go to leftmost node and print

Threaded tree Transversal

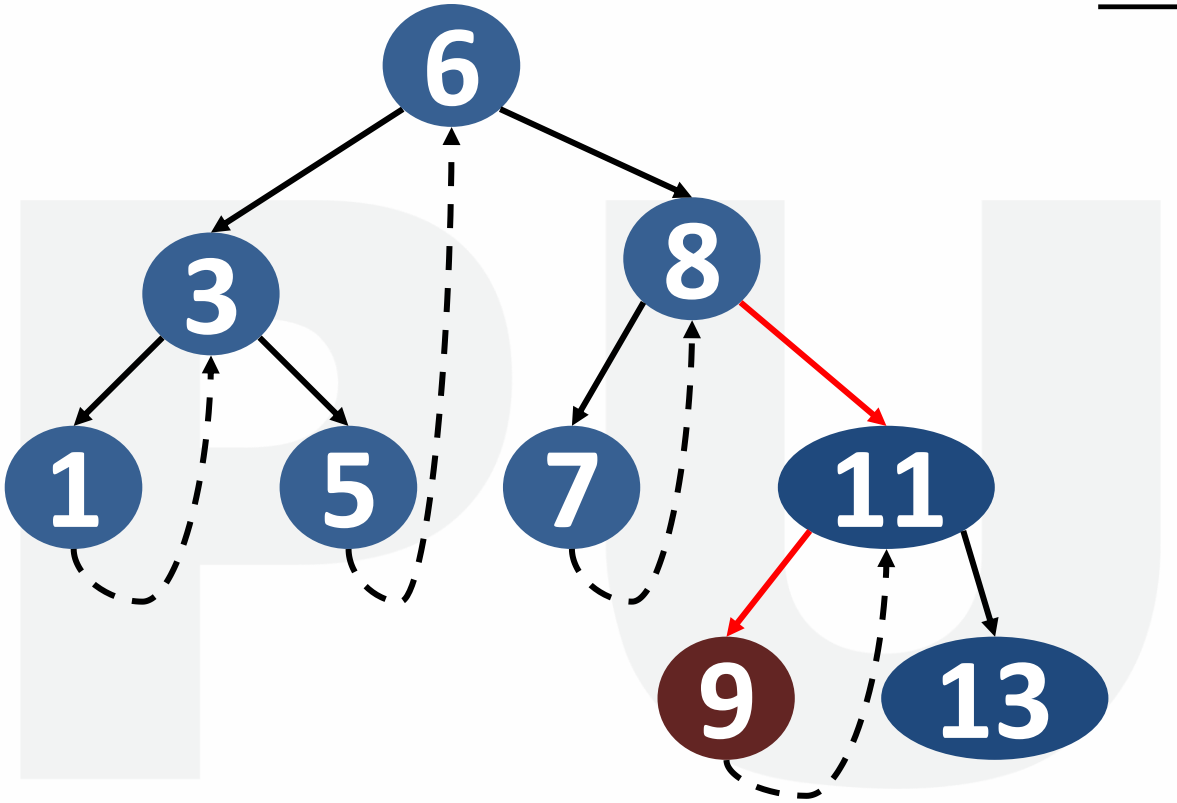


Output

1
3
5
6
7
8

Follow thread to right, print node

Threaded tree Transversal

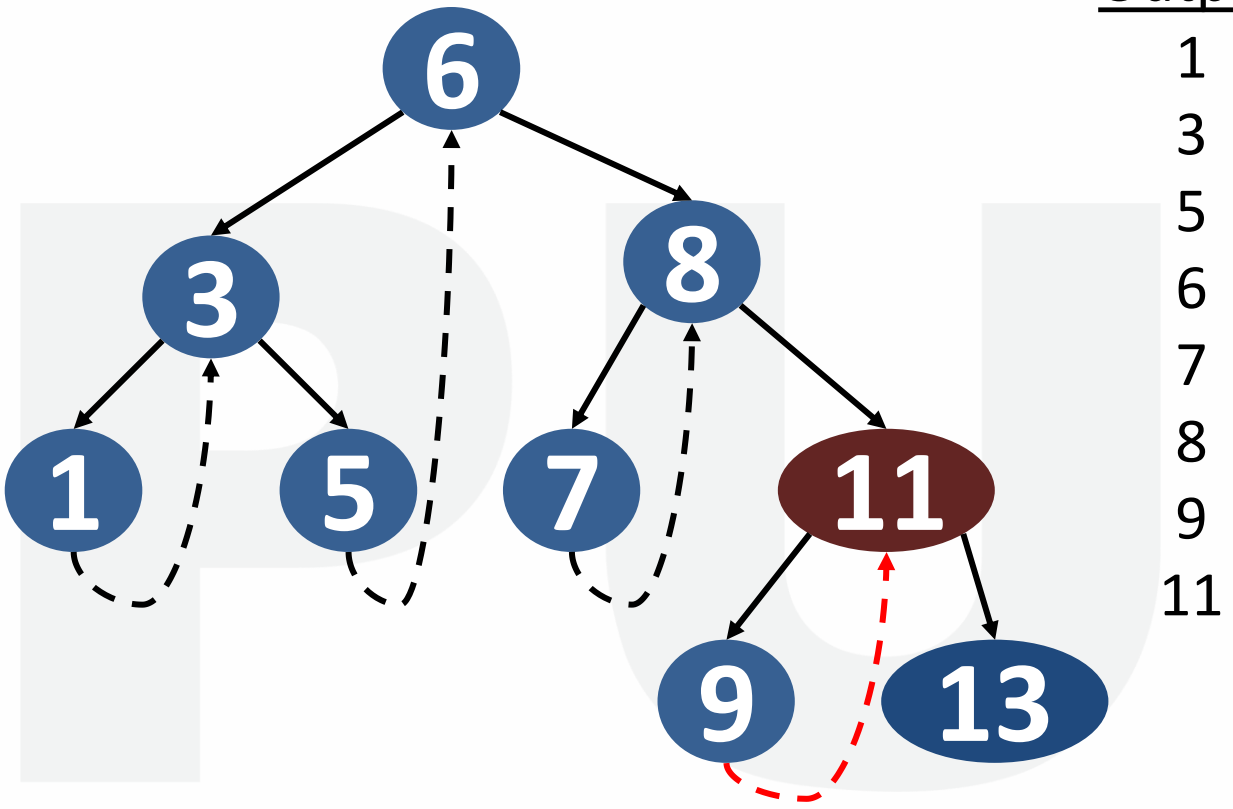


Output

1
3
5
6
7
8
9

Follow link to right, go to leftmost node and print

Threaded tree Transversal

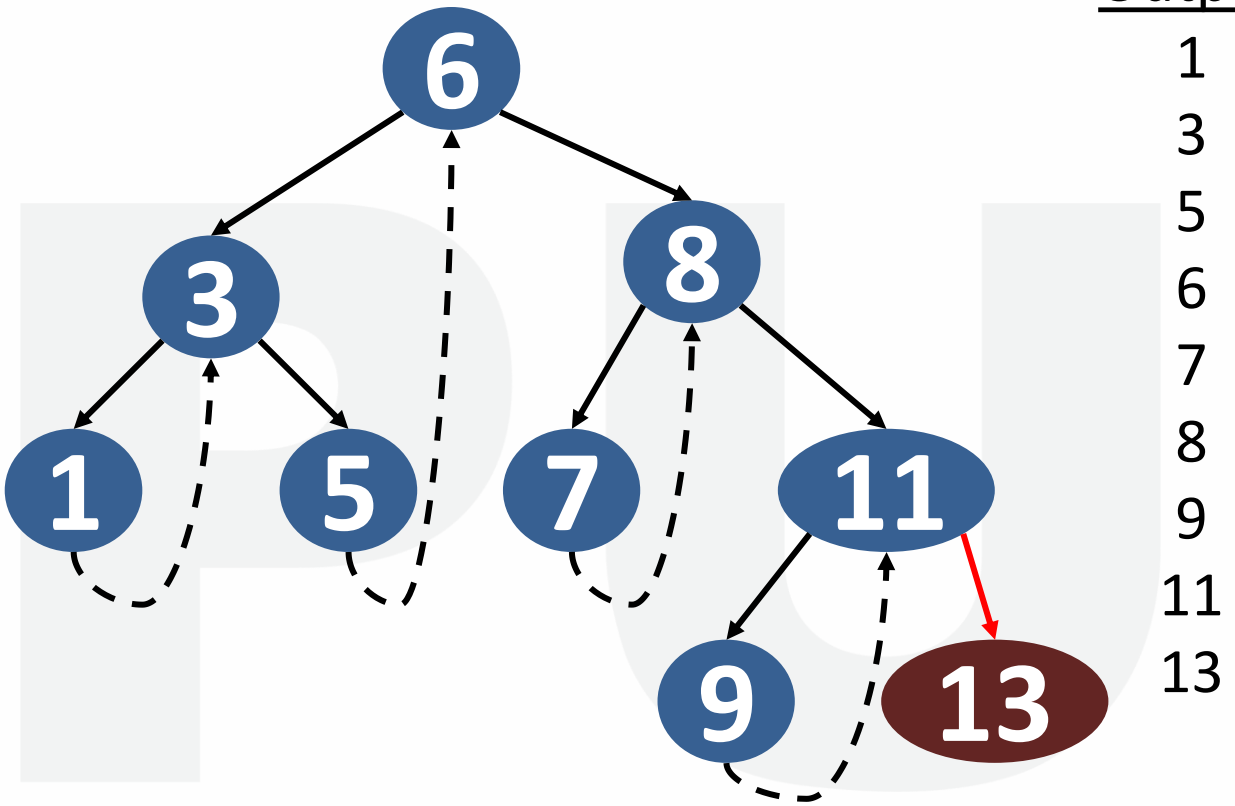


Output

1
3
5
6
7
8
9
11

Follow thread to right, print node

Threaded tree Transversal



Output

1
3
5
6
7
8
9
11
13

Follow link to right, go to leftmost node and print

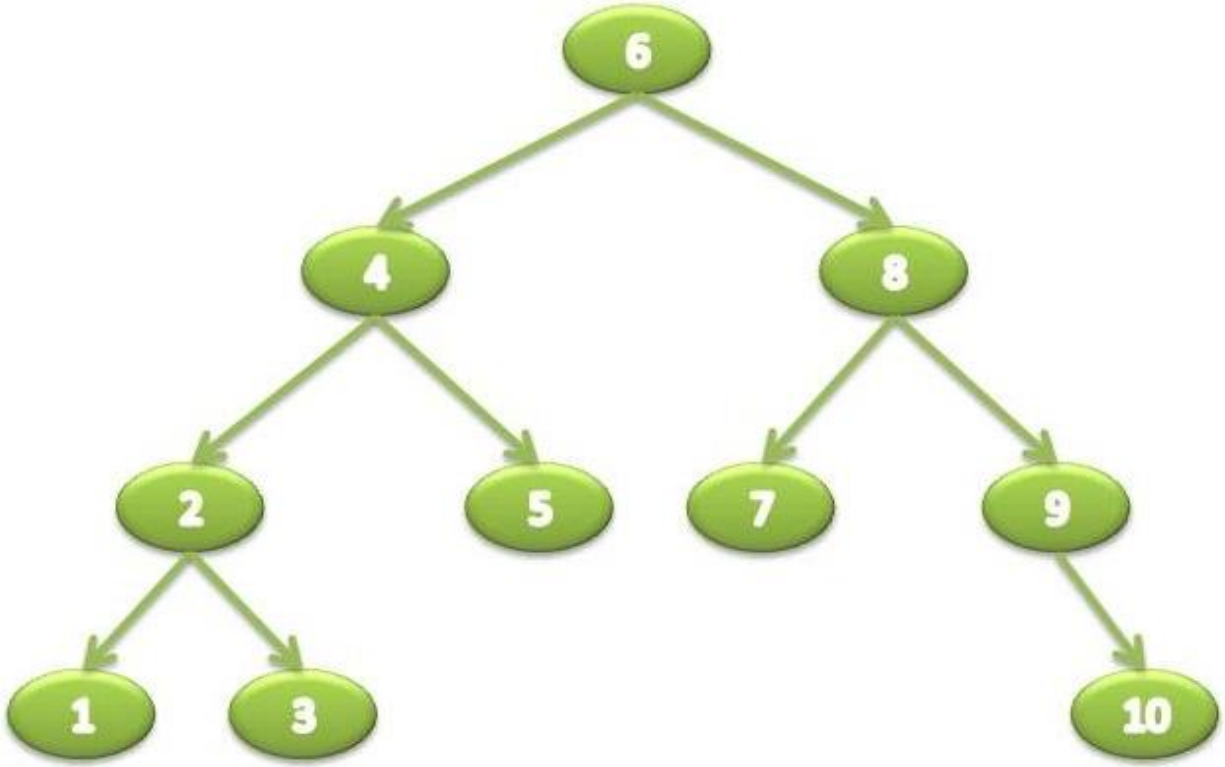
Advantage of Threaded Binary tree

- The traversal operation is more faster than that of its unthreaded version.
- Because with threaded binary tree non-recursive implementation is possible which can run faster and does not require the botheration of stack management.
- we can efficiently determine the predecessor and successor nodes starting from any node.

Disadvantage of Threaded Binary tree

- Insertion and deletion from a threaded tree are very time consuming operation compare to non-threaded binary tree.
- This tree require additional bit to identify the threaded link.

Binary Search Tree



Binary Search Tree

- The node in extreme left will be the smallest node in the tree.
- The node in the extreme right will be the largest node in the tree.
- Since nodes in binary search trees are ordered the time needed to search an element is greatly reduced.

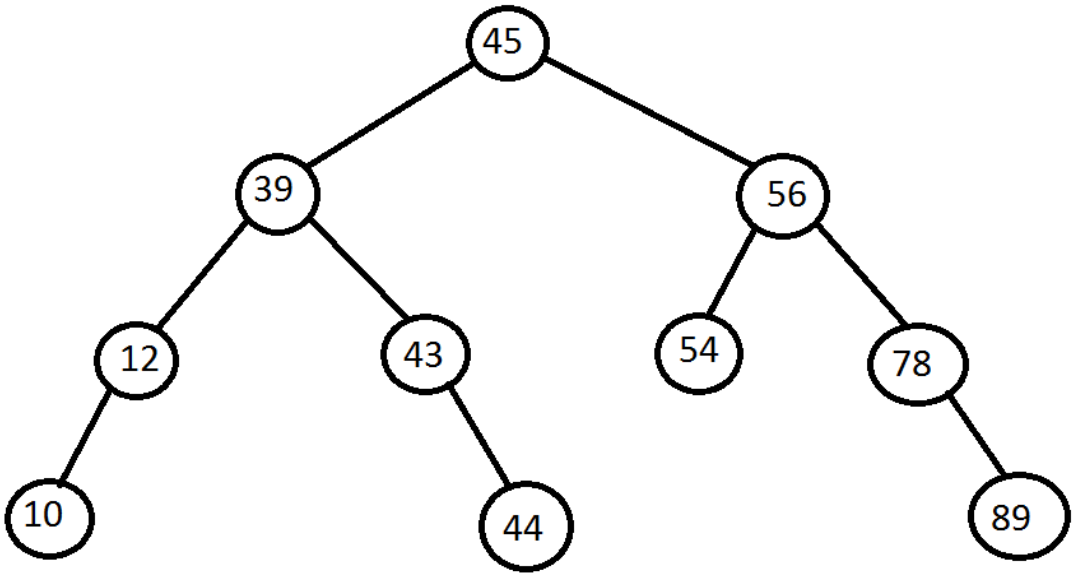
Operation on Binary Search Tree

- Search
- Insertion
- Deletion

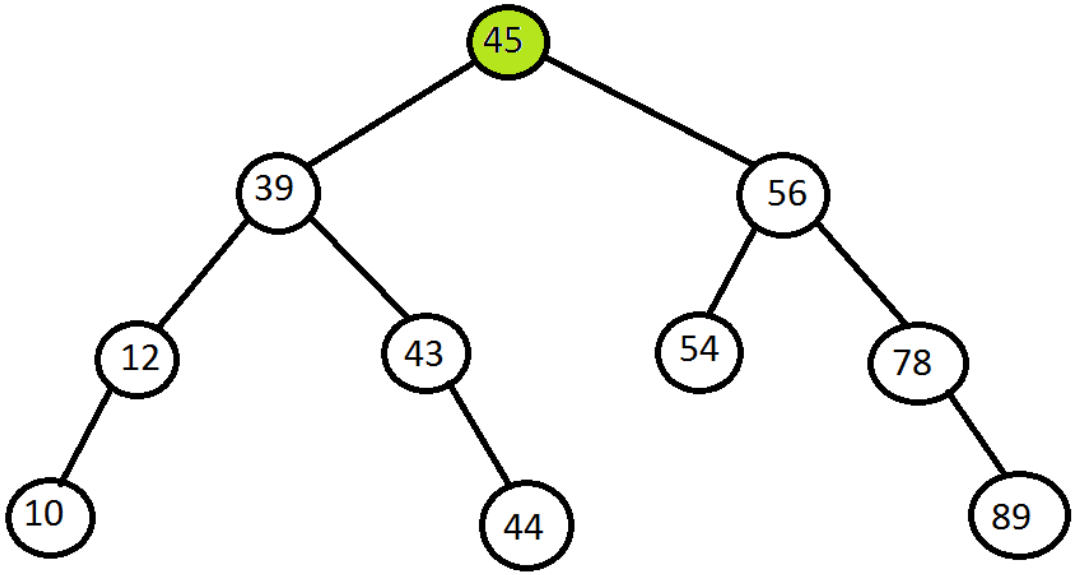
Search

- Search function is used to find whether a value is present in a tree or not.
- This process begins at the root node.
- The function first checks if the binary tree is empty.
- Then it checks if the value to be searched is less than or greater than the node.
- If the value is less than it will go to the left sub tree else to the right sub tree.

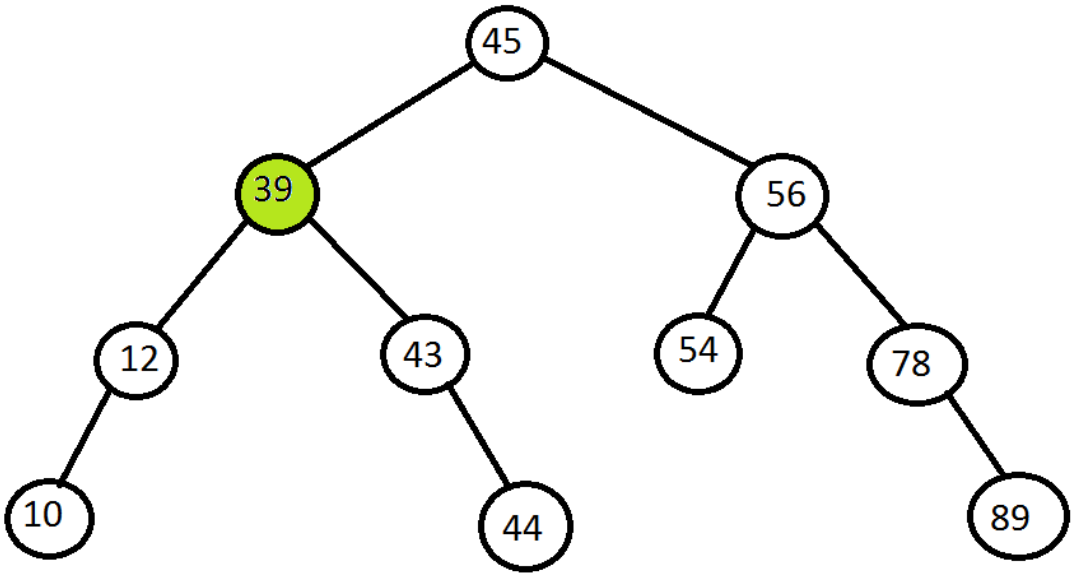
Search node 44



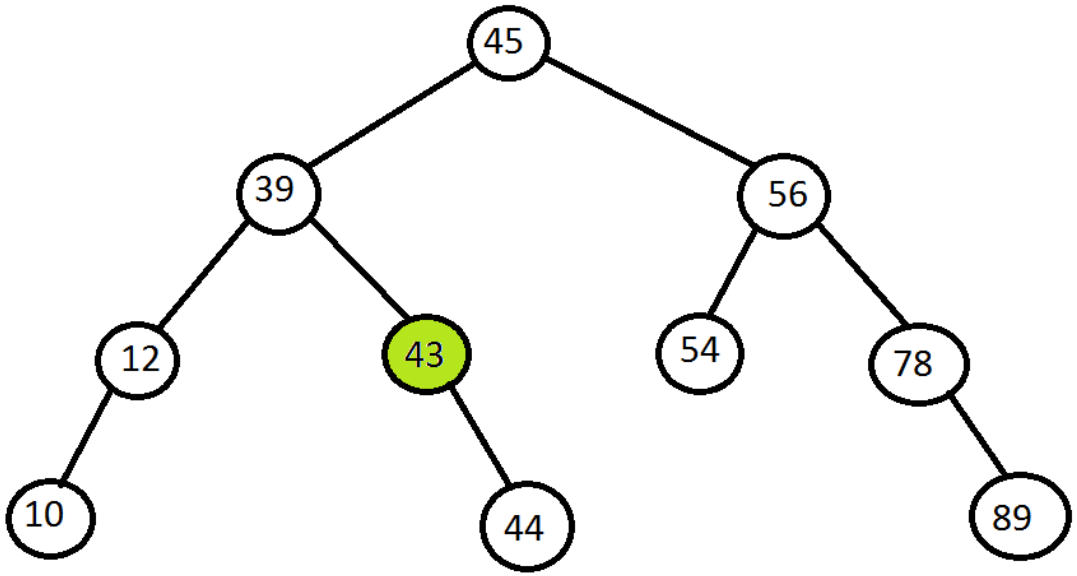
Search node 44



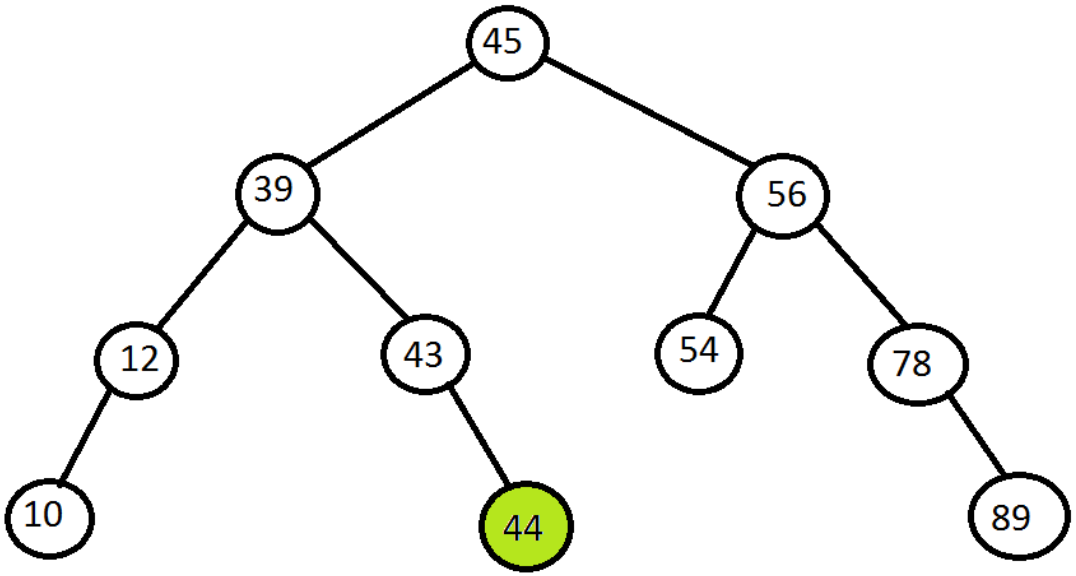
Search node 44



Search node 44



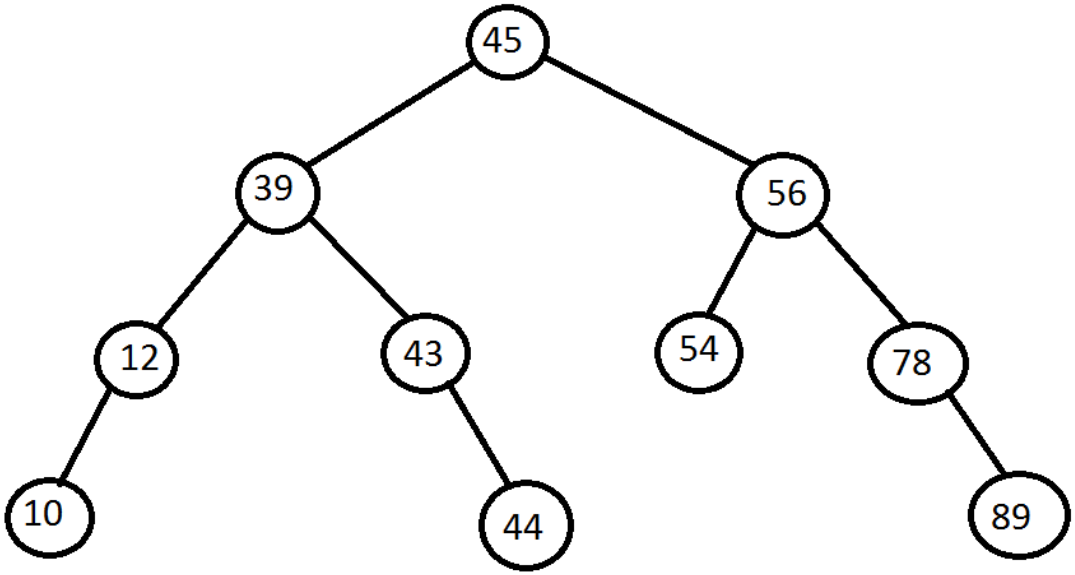
Search node 44



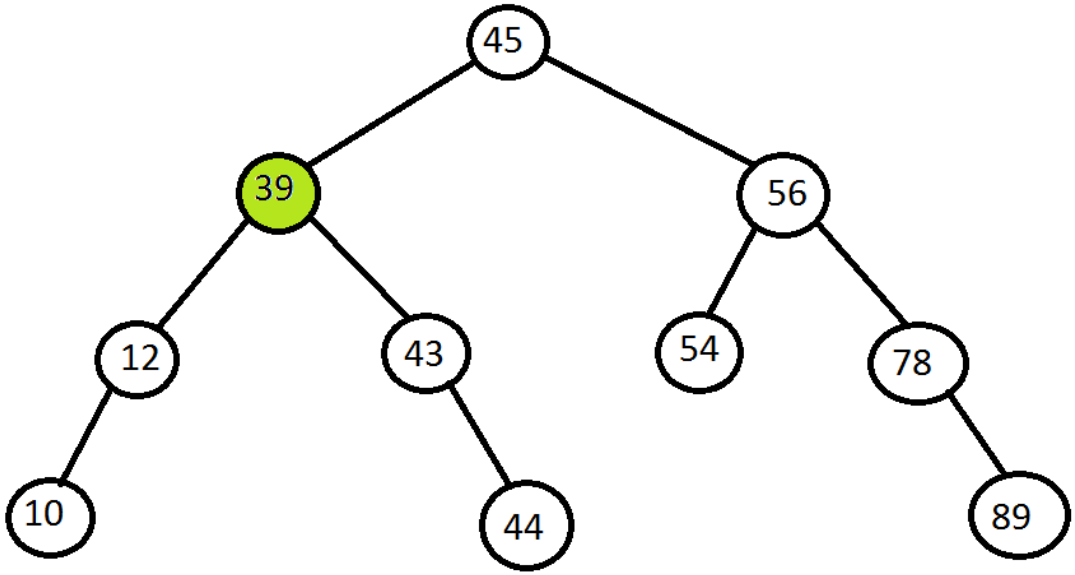
Insert

- Insert is used to add a new node with a given value at the correct position in the binary search tree.
- The insert function continues moving down the tree until it reaches a leaf node.
- If the new node’s value is greater than the parent node then the new node is inserted in right sub tree else in the left sub tree.

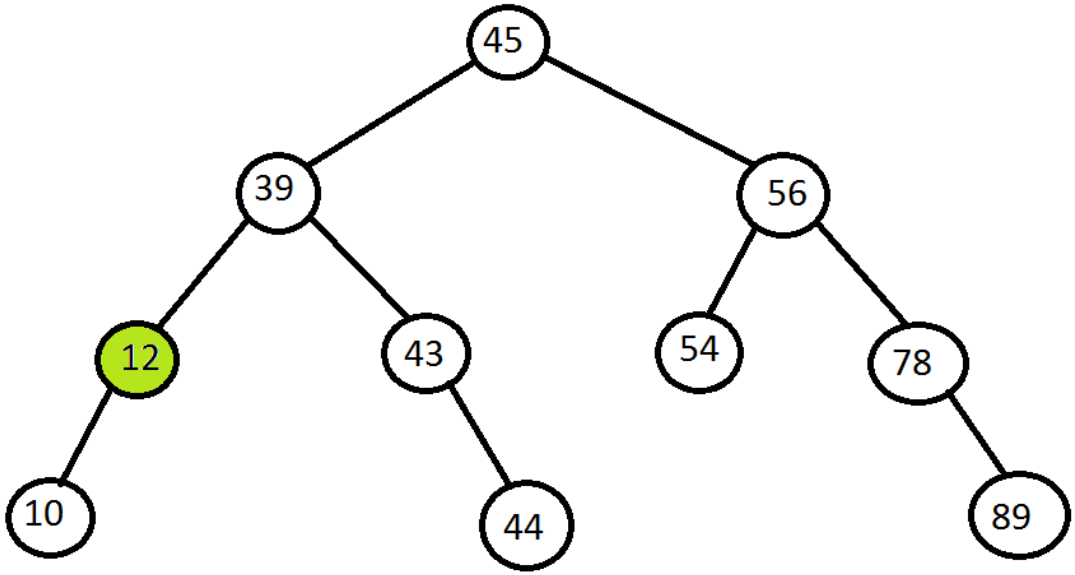
Insert Value 20



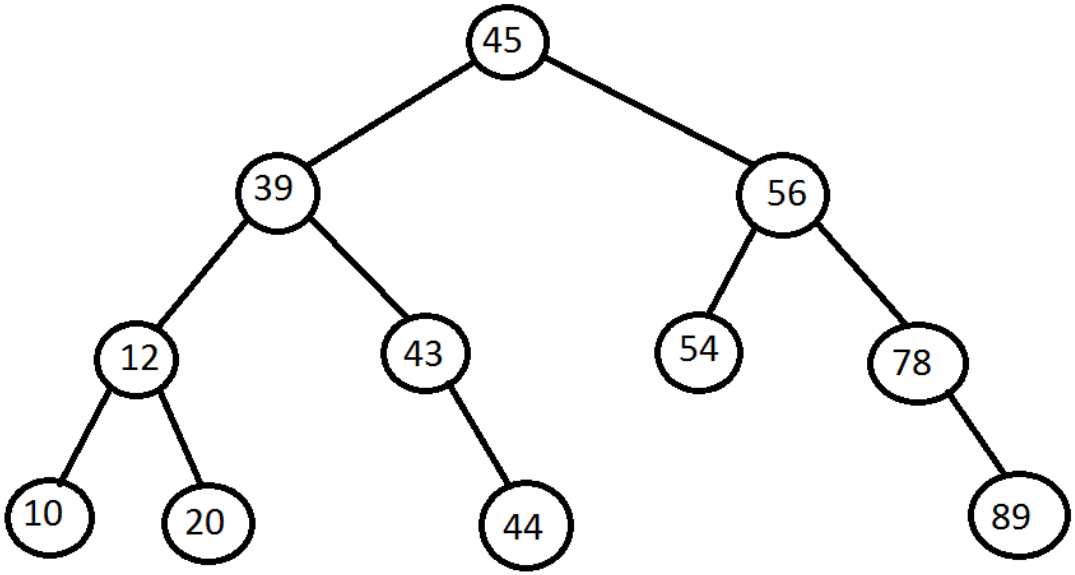
Insert Value 20



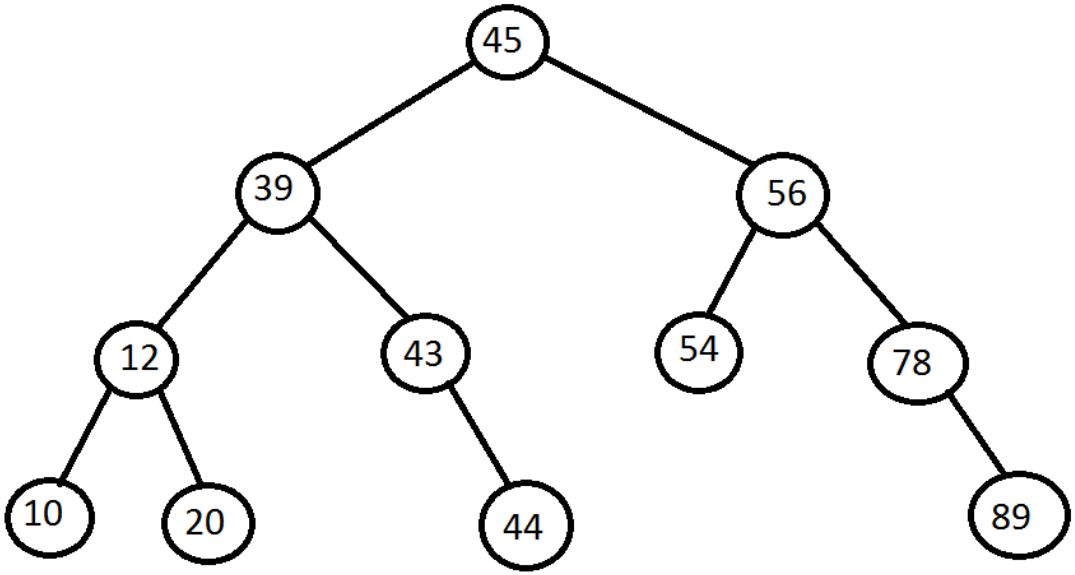
Insert Value 20



Insert Value 20



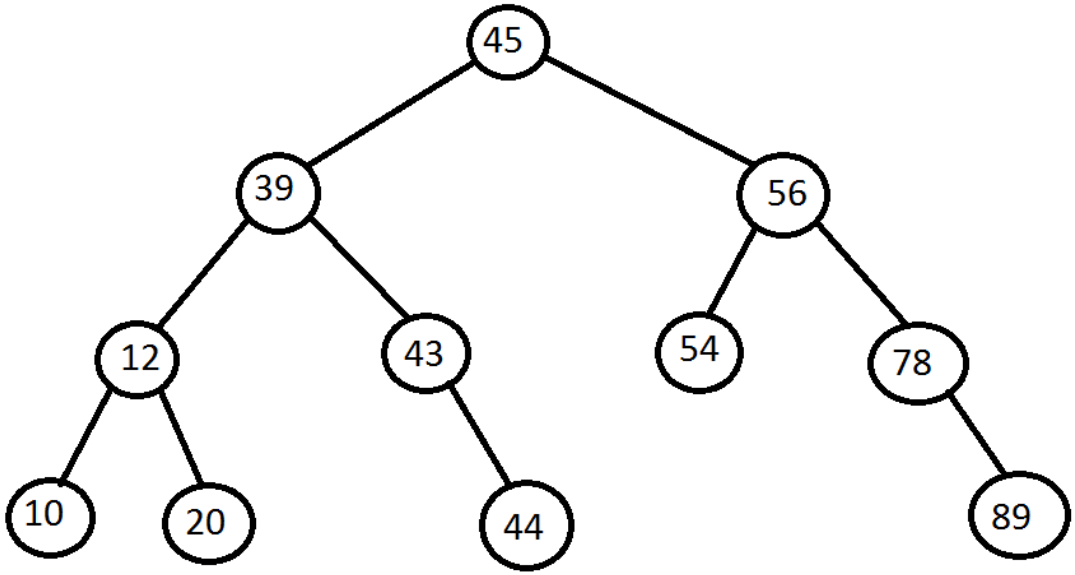
Insert Value 20



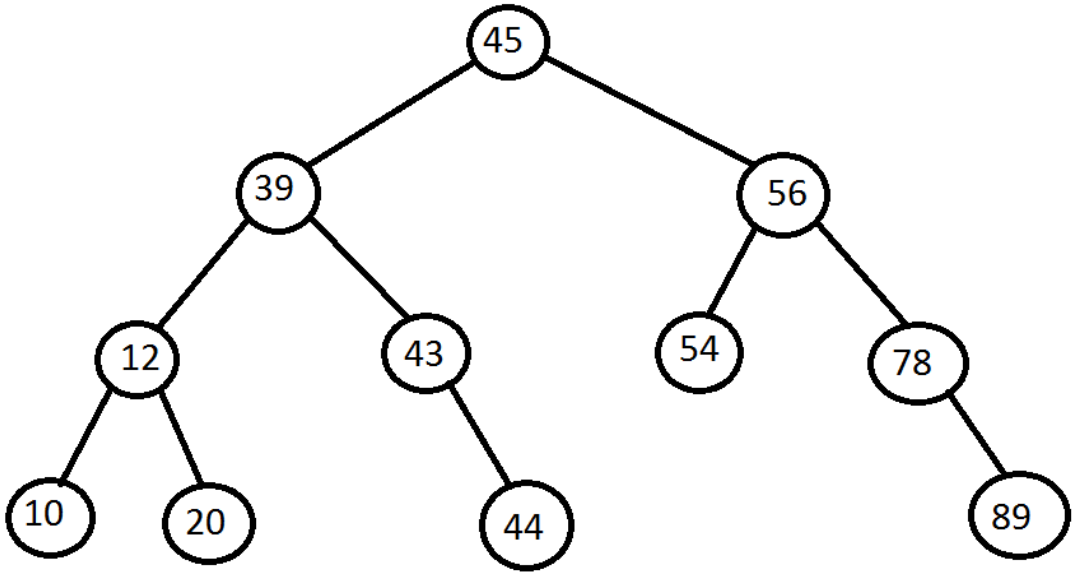
Deletion

1) Deleting a node that has no children.
2) Deletion a node that has one child.
3) Deletion a node that has two children.

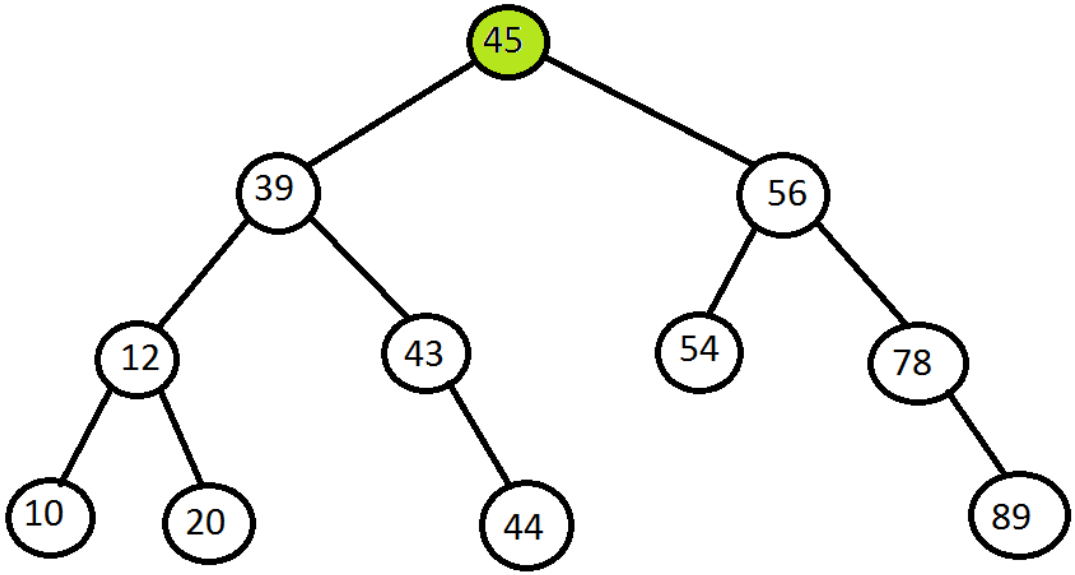
Deletion



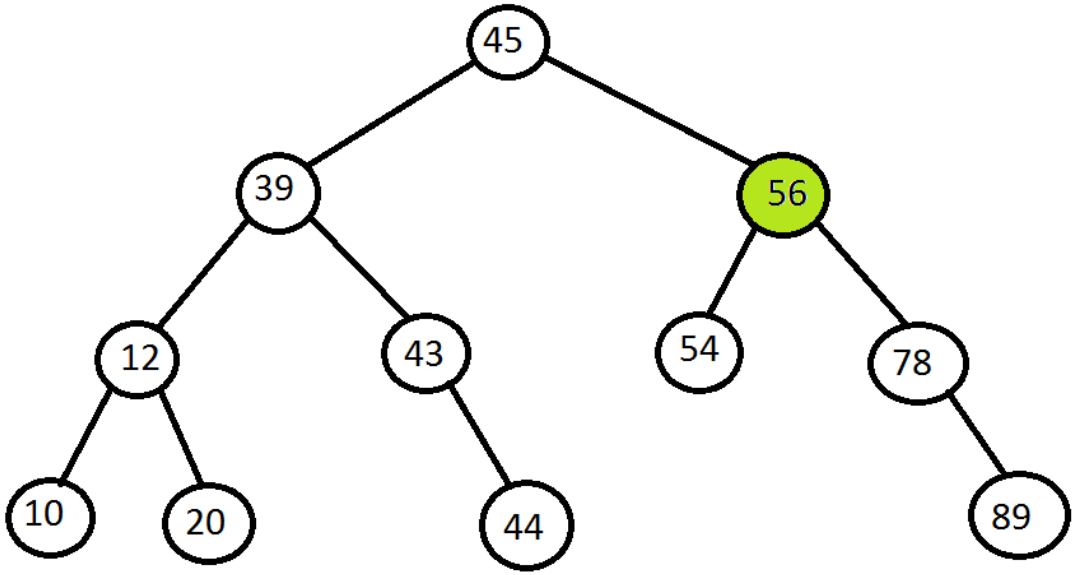
Deletion node 54



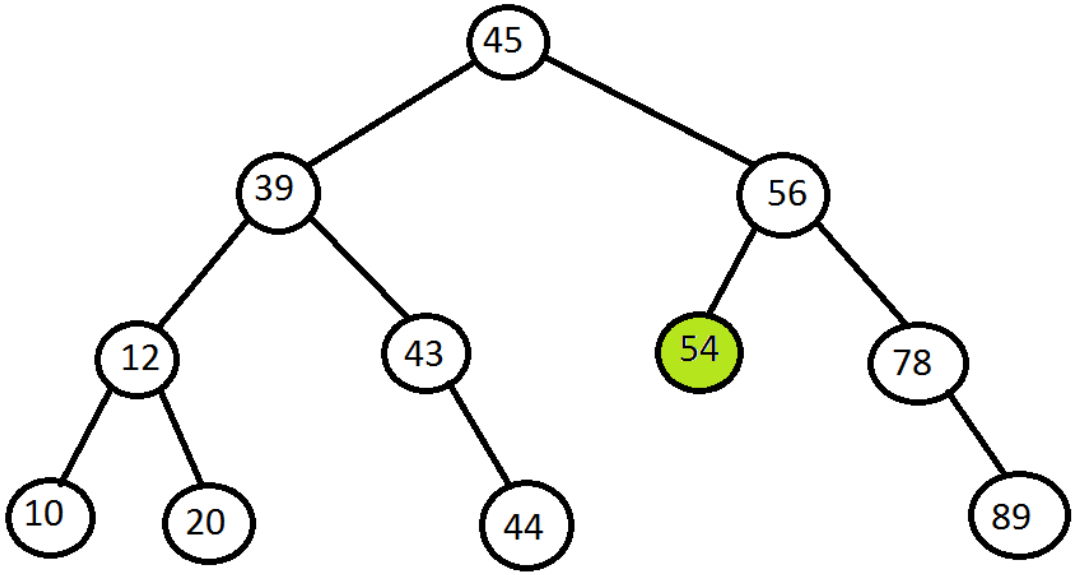
Deletion node 54



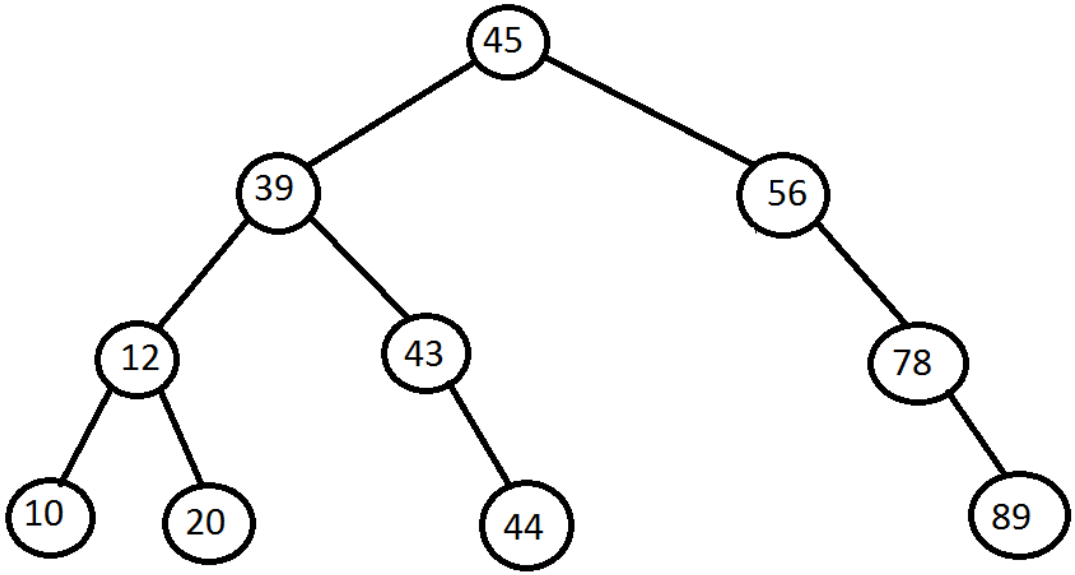
Deletion node 54



Deletion node 54



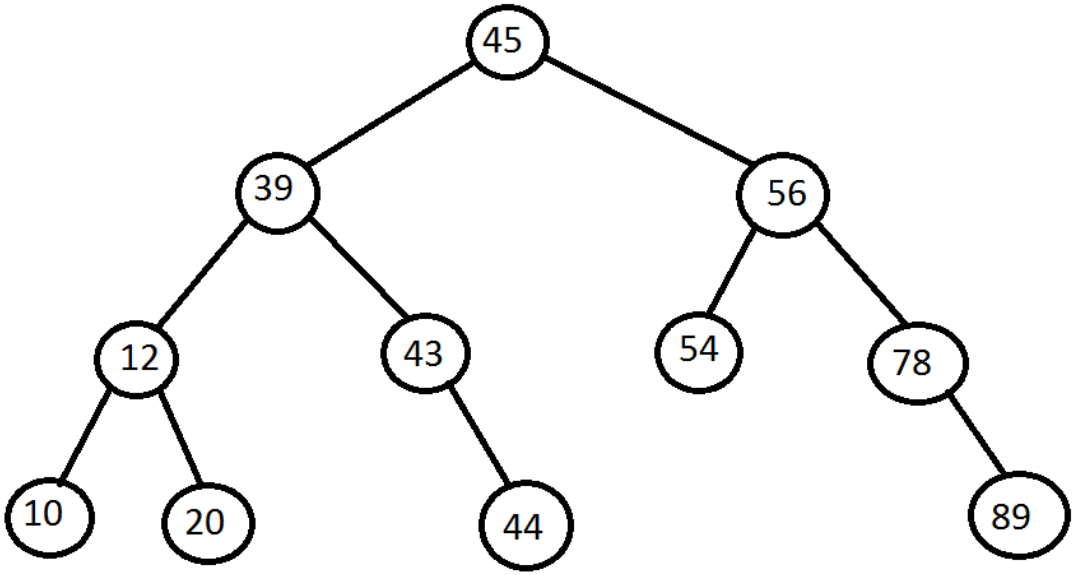
Deletion node 54



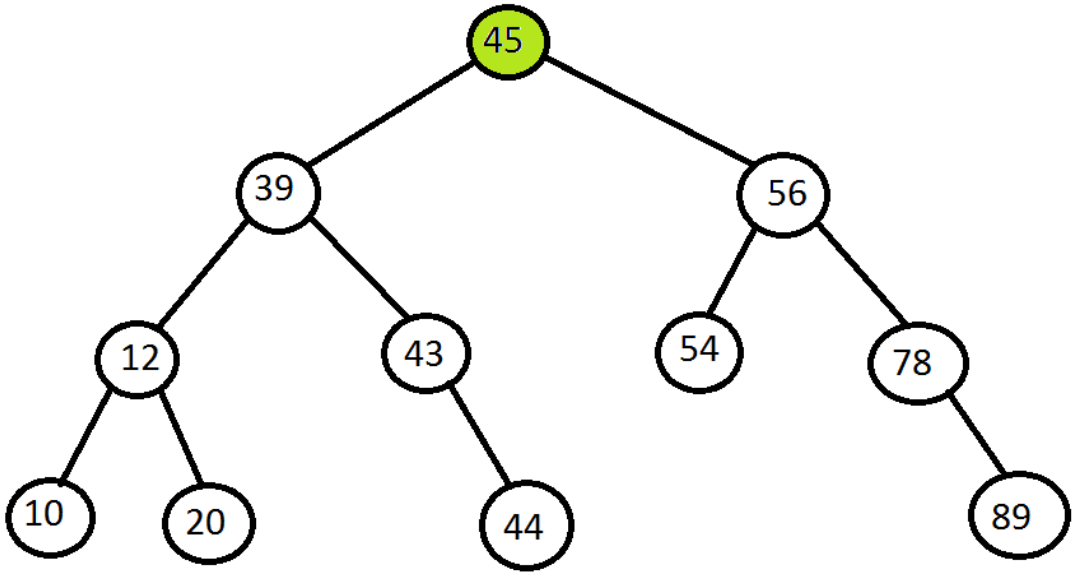
Deletion a node with one child

- To handle this case node's child is set to be child of node's parent.
- If node was left child of its parent then node's child becomes left child of node's parent.
- If node was right child of it's parent then node's child becomes right child of node's parent.

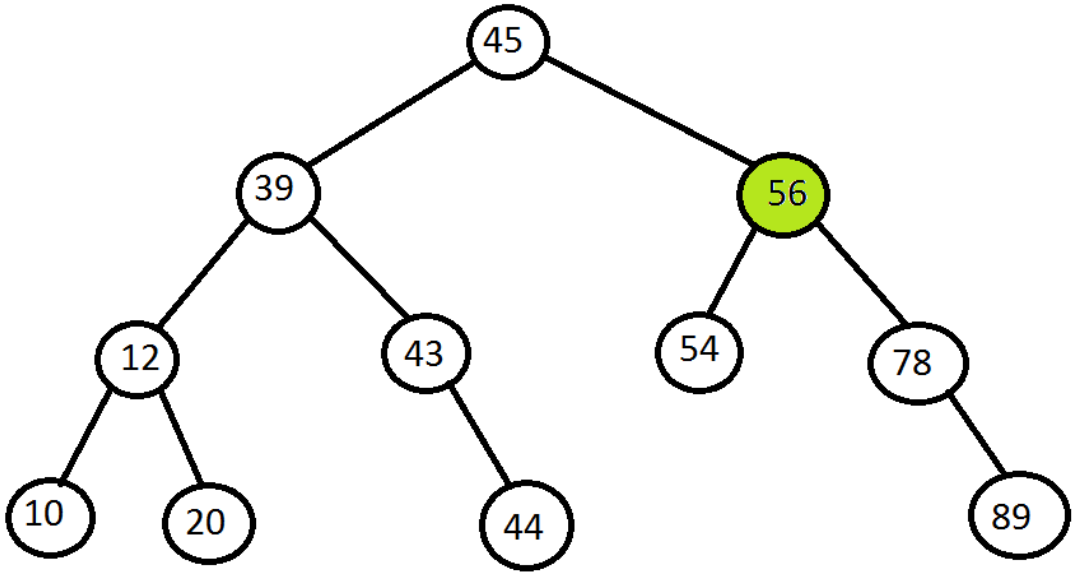
Deletion a node 78



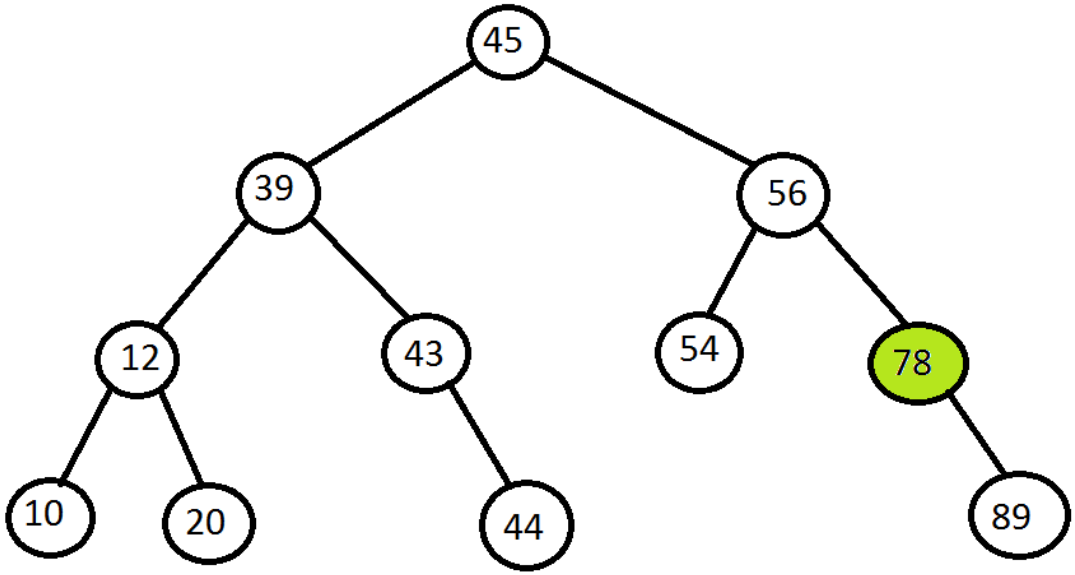
Deletion a node 78



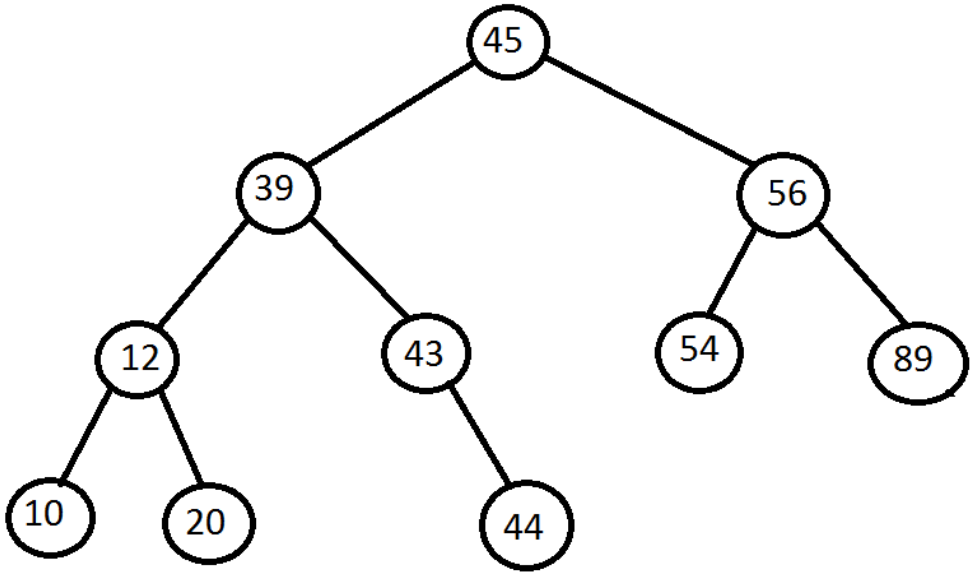
Deletion a node 78



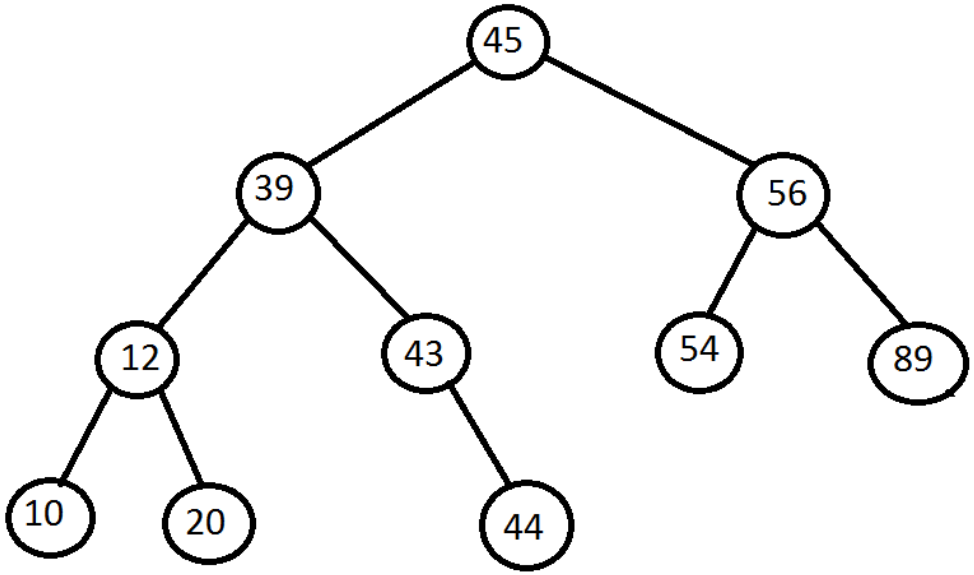
Deletion a node 78



Deletion a node 78



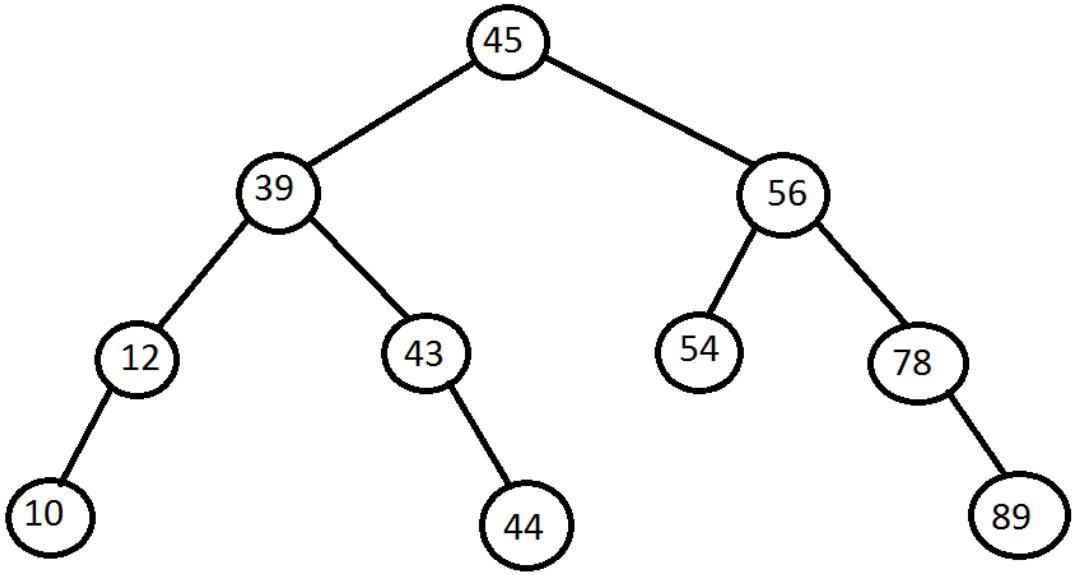
Deletion a node 78



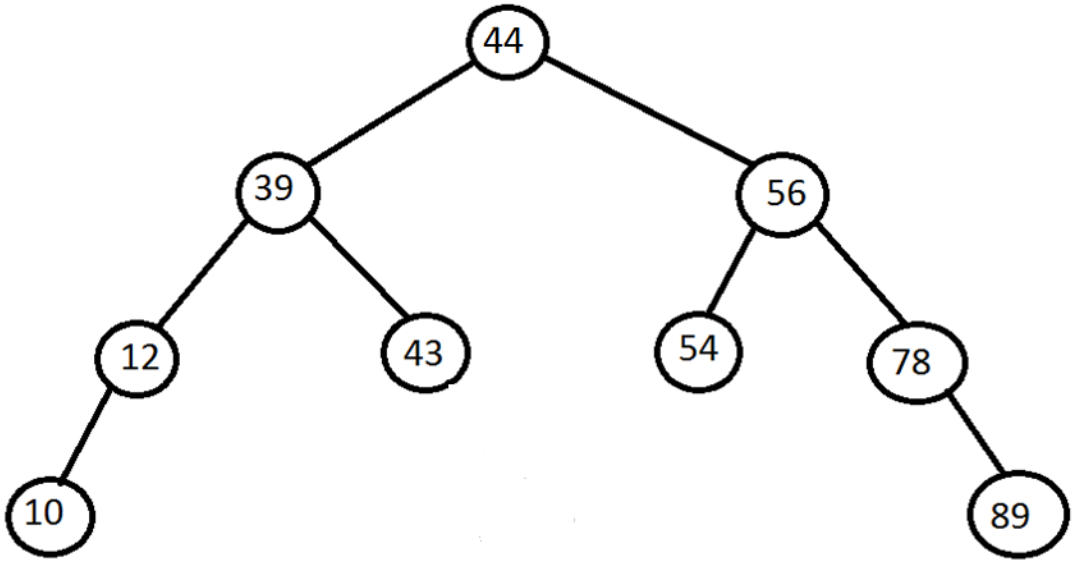
Deletion a node with Two child

- To handle this case replace node's value with its **IN ORDER PREDECESSOR** or **IN ORDER SUCCESSOR.**
- In order predecessor is right most child of left sub tree.
- In order successor is left most child of right sub tree.

Deletion a node with 45



Deletion a node with 45



Parul® University

UNIT 4
Tree

Prof. Pintu Chauhan, Assistant Professor
Computer Science & Engineering

Parul®
University

DIGITAL LEARNING CONTENT

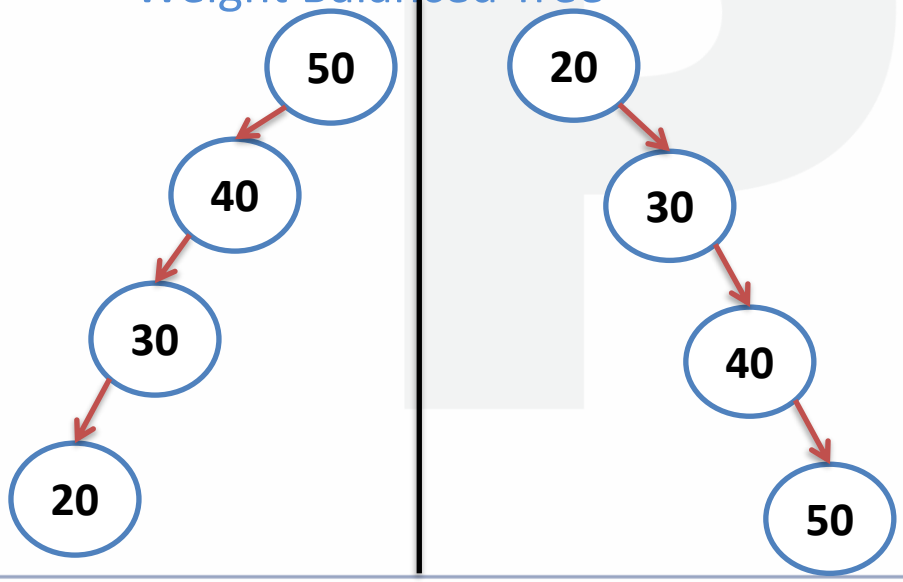
Topic-2

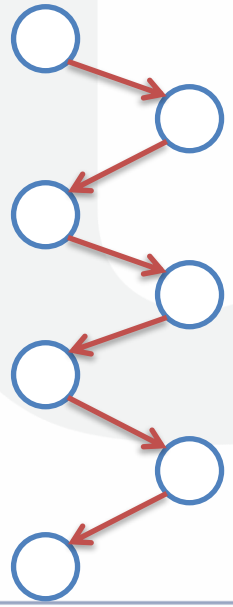
AVL Tree

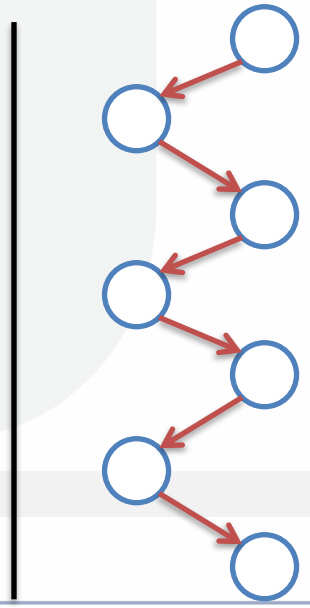
Balanced Tree

- Binary Search Tree gives advantage of Fast Search, but sometimes in few cases we are not able to get this advantage. E.g. look into worst case BST
- Balanced binary trees are classified into two categories
  - Height Balanced Tree (AVL Tree)
  - Weight Balanced Tree

Worst search time cases for Binary Search Tree



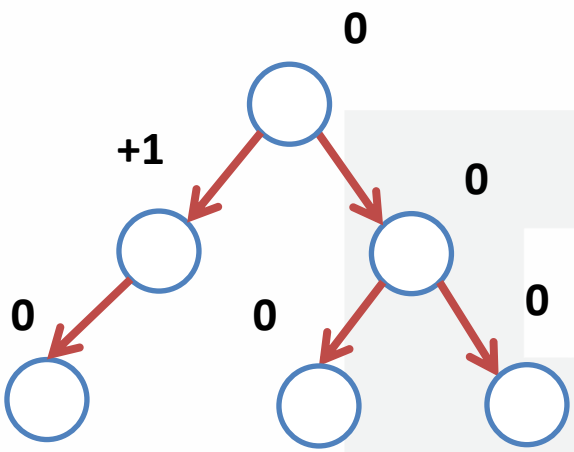


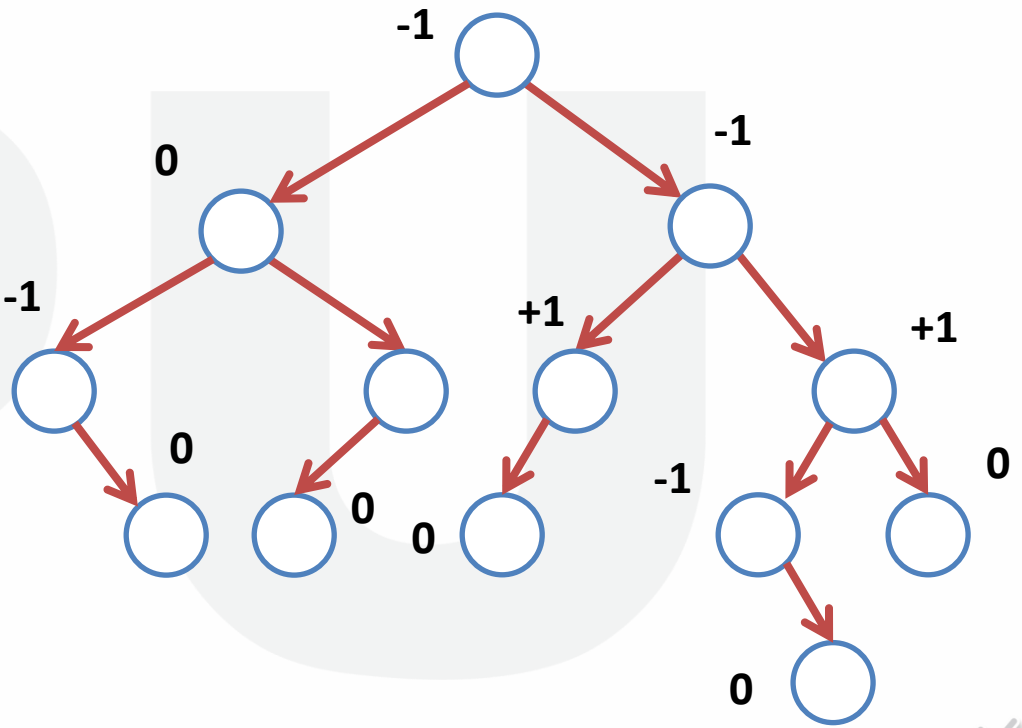


Height Balanced Tree (AVL Tree)

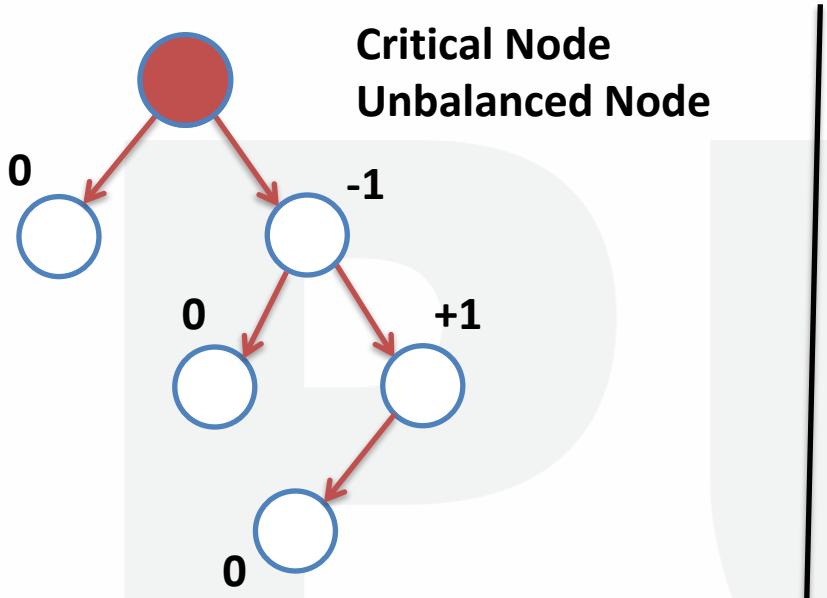
- A Balance factor of a node $|H_L - H_R| <= 1$, where H_L, H_R are heights of left and right sub-tree respectively
- Permissible balance factors:
  - A Left-High (balance factor +1)
    The left-sub tree is one level taller than the right-sub tree
  - Balanced (balance factor 0)
    The left and right sub-trees are both the same heights
  - Right-High (balance factor -1)
    The right sub-tree is one level taller than the left-sub tree.
- In height balanced tree, each node must be in one of these states
- If there exists a node in a tree where this is not true, then such a tree is called **Unbalanced**

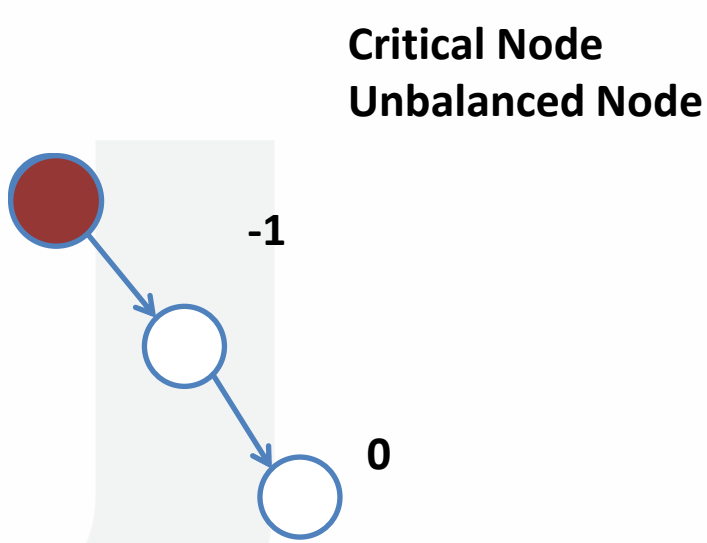
AVL Tree





AVL Tree





- Sometimes tree becomes unbalanced by inserting or deleting any node
- Then based on position of insertion, we need to rotate the unbalanced node
- **Rotation** is the **process** to **make tree balanced**

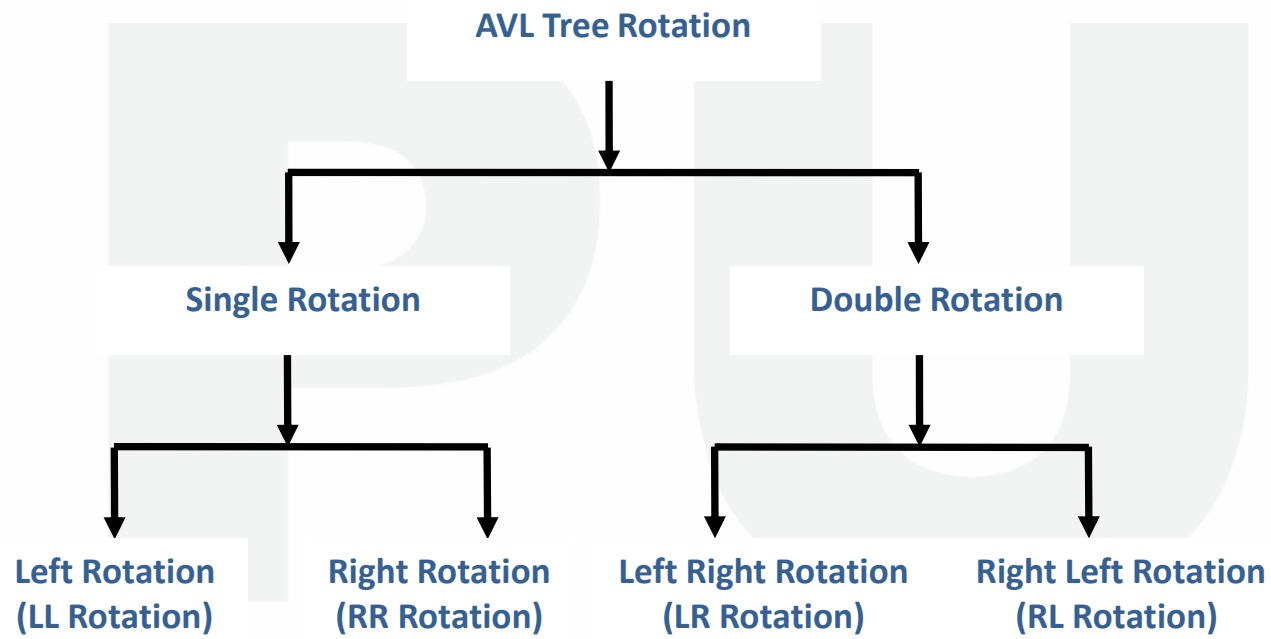
AVL Tree Rotation

The following operations are performed on AVL tree...

- **Search**
- **Insertion**
- **Deletion**

AVL Tree Rotation

There are **four** rotations and they are classified into **two** types.



Operation on an AVL

- In AVL tree, after performing operations like insertion and deletion we need to check the **balance factor** of every node in the tree. Permissible balance factors:
- If every node satisfies the balance factor condition then we conclude the operation otherwise we must make it balanced.
- If there exists a node in a tree where this is not true, then such a tree is called **Unbalanced**

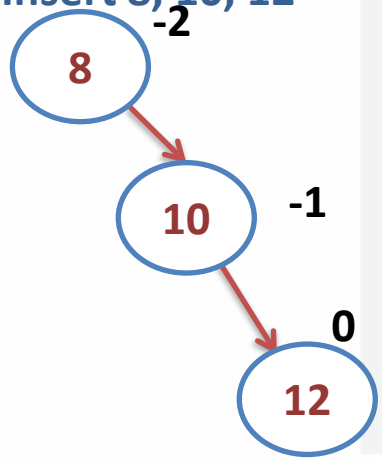
Rotation operations are used to make the tree balanced.

Rotation is the process of moving nodes either to left or to right to make the tree balanced.

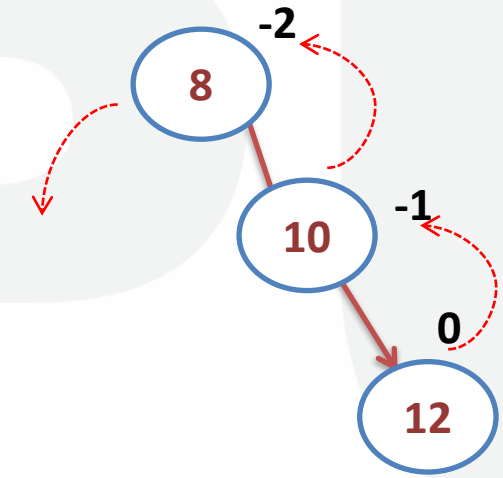
Single Left Rotation (LL Rotation)

- In LL Rotation, every node moves one position to left from the current position.
- To understand LL Rotation, let us consider the following insertion operation in AVL Tree.

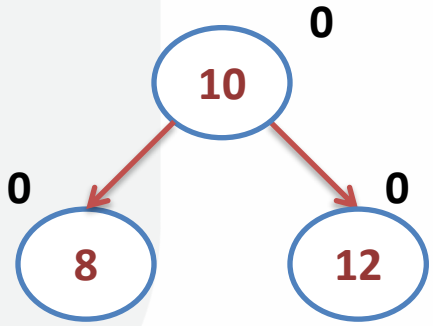
Insert 8, 10, 12



Tree is imbalanced



To make balanced we use LL Rotation which moves nodes one position to left

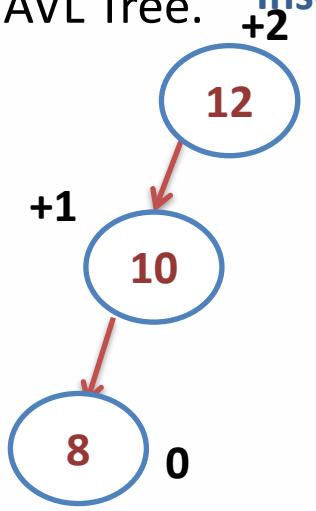


After LL Rotation Tree is balanced

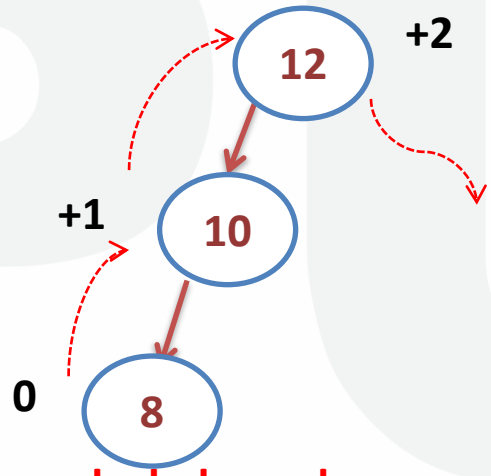
Single Right Rotation (RR Rotation)

- In RR Rotation, every node moves one position to right from the current position.
- To understand RR Rotation, let us consider the following insertion operation in AVL Tree.

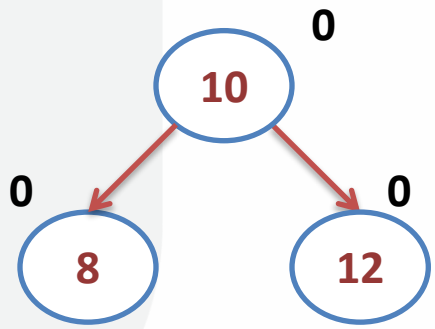
Insert 12, 10, 8



Tree is imbalanced



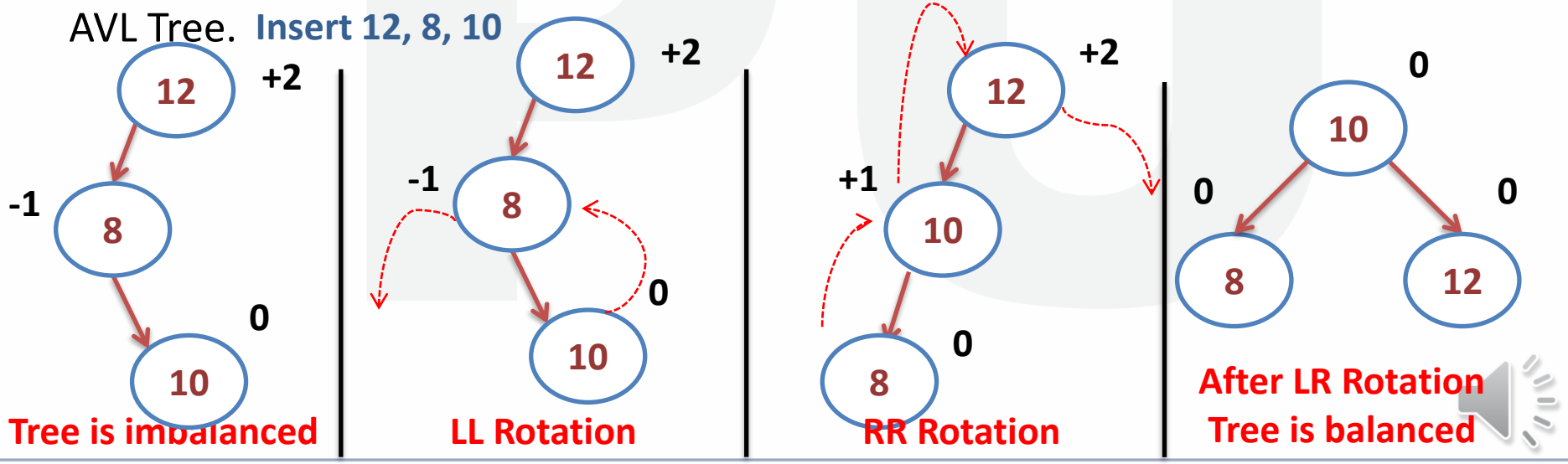
To make balanced we use RR Rotation which moves nodes one position to Right



After RR Rotation Tree is balanced

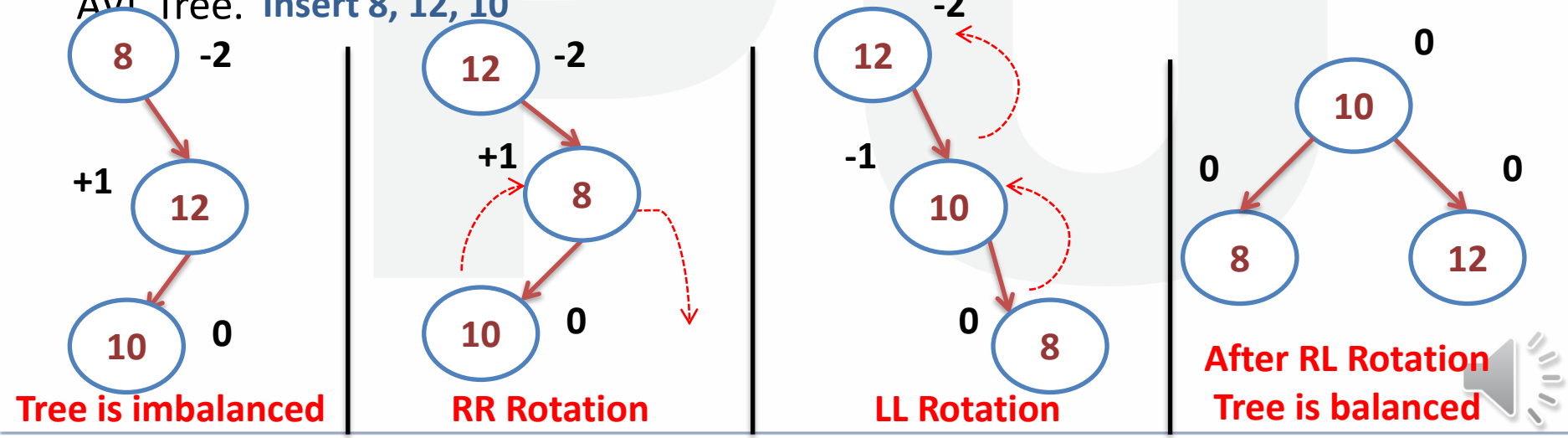
Left Right Rotation (LR Rotation)

- The LR Rotation is a sequence of single left rotation followed by a single right rotation
- In LR Rotation, at first, every node moves one position to the left and one position to right from the current position.
- To understand LR Rotation, let us consider the following insertion operation in AVL Tree. **Insert 12, 8, 10**



Right Left Rotation (LR Rotation)

- The RL Rotation is a sequence of single right rotation followed by a single left rotation
- In RL Rotation, at first, every node moves one position to the right and one position to left from the current position.
- To understand RL Rotation, let us consider the following insertion operation in AVL Tree. **Insert 8, 12, 10**



AVL Tree Rotation

The following operations are performed on AVL tree...

- **Search**
- **Insertion**
- **Deletion**

Search Operation in AVL Tree

- In an AVL tree, the search operation is performed with **O(log n)** time complexity.
- The search operation in the AVL tree is similar to the search operation in a Binary search tree.
- We use the following steps to search an element in AVL tree.

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 matched, then display "Given node is found!!!" and terminate the function

Step 4 - If both are not matched, then check whether search element is smaller or larger than that node value.

Search Operation in AVL Tree

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 find the exact element or until the search element is compared with the leaf node.

Step 8 - If we reach to the node having the value equal to the search value, then display "Element is found" and terminate the function.

Step 9 - If we reach to the leaf node and if it is also not matched with the search element, then display "Element is not found" and terminate the function.

Insertion Operation in AVL Tree

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

Step 1 - Insert the new element into the tree using Binary Search Tree insertion logic.

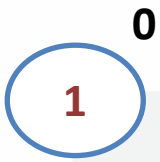
Step 2 - After insertion, check the **Balance Factor** of every node.

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 that tree is said to be imbalanced. In this case, perform suitable **Rotation** to make it balanced and go for next operation.

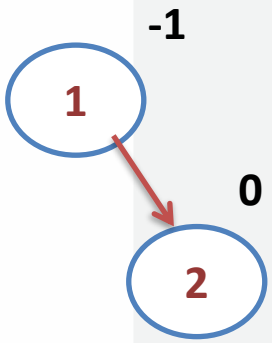
Construct an AVL tree by inserting numbers from 1 to 7

Insert 1



Tree is balanced

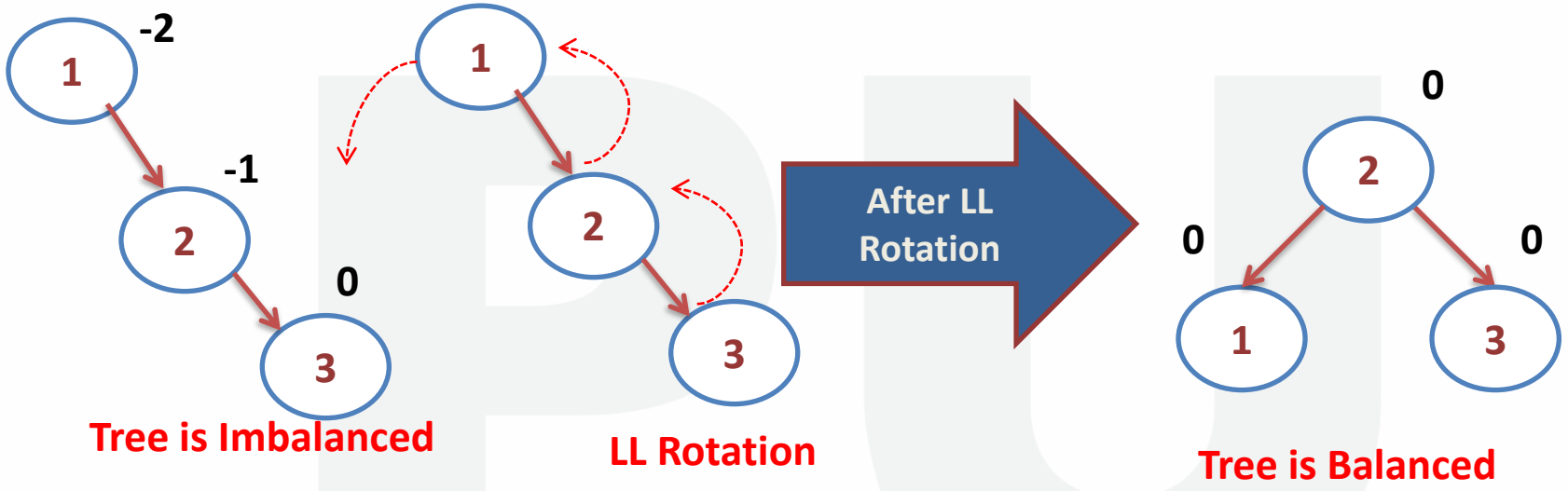
Insert 2



Tree is balanced

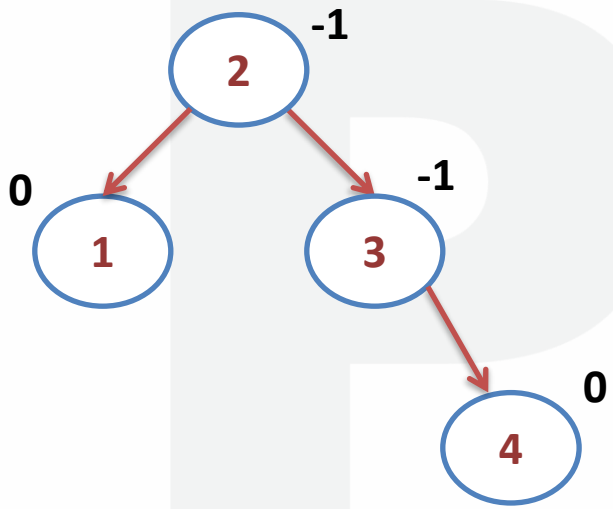
Construct an AVL tree by inserting numbers from 1 to 7

Insert 3



Construct an AVL tree by inserting numbers from 1 to 7

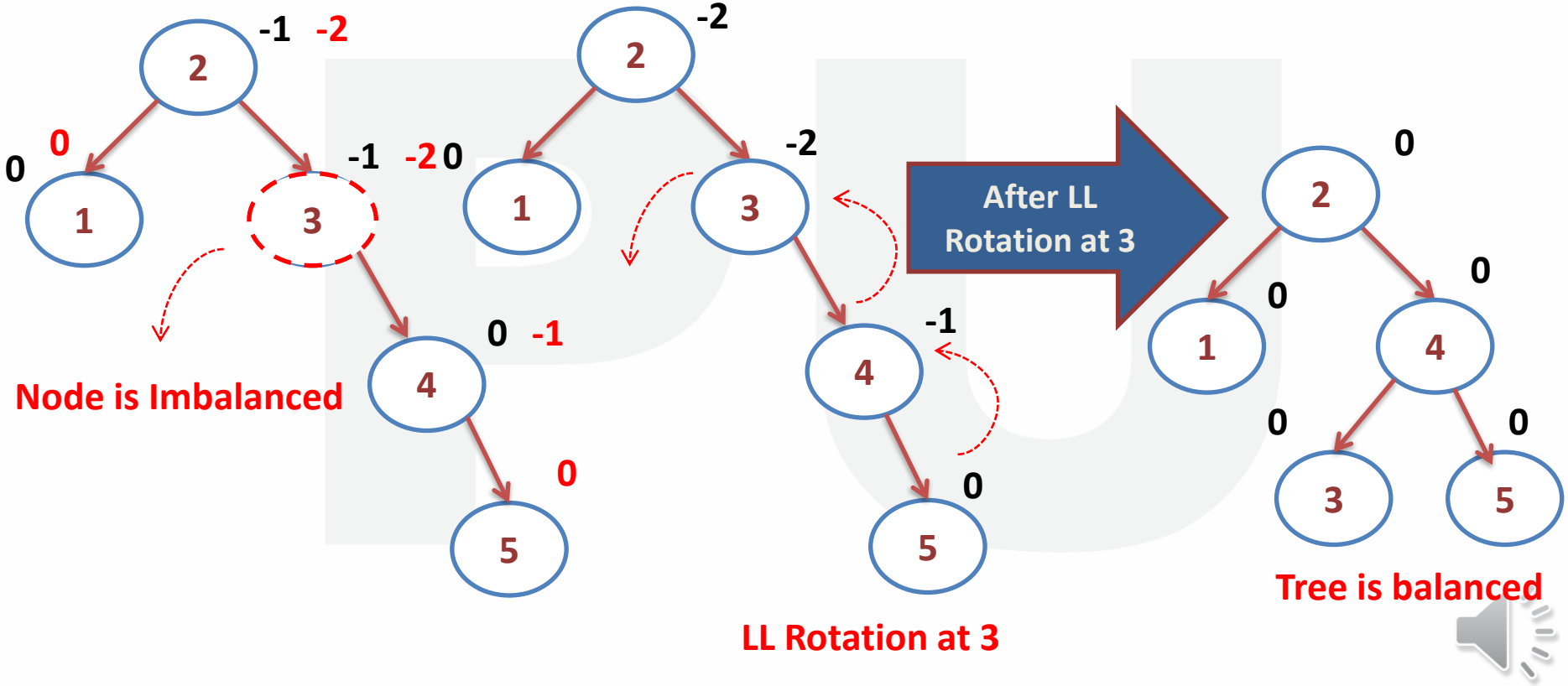
Insert 4



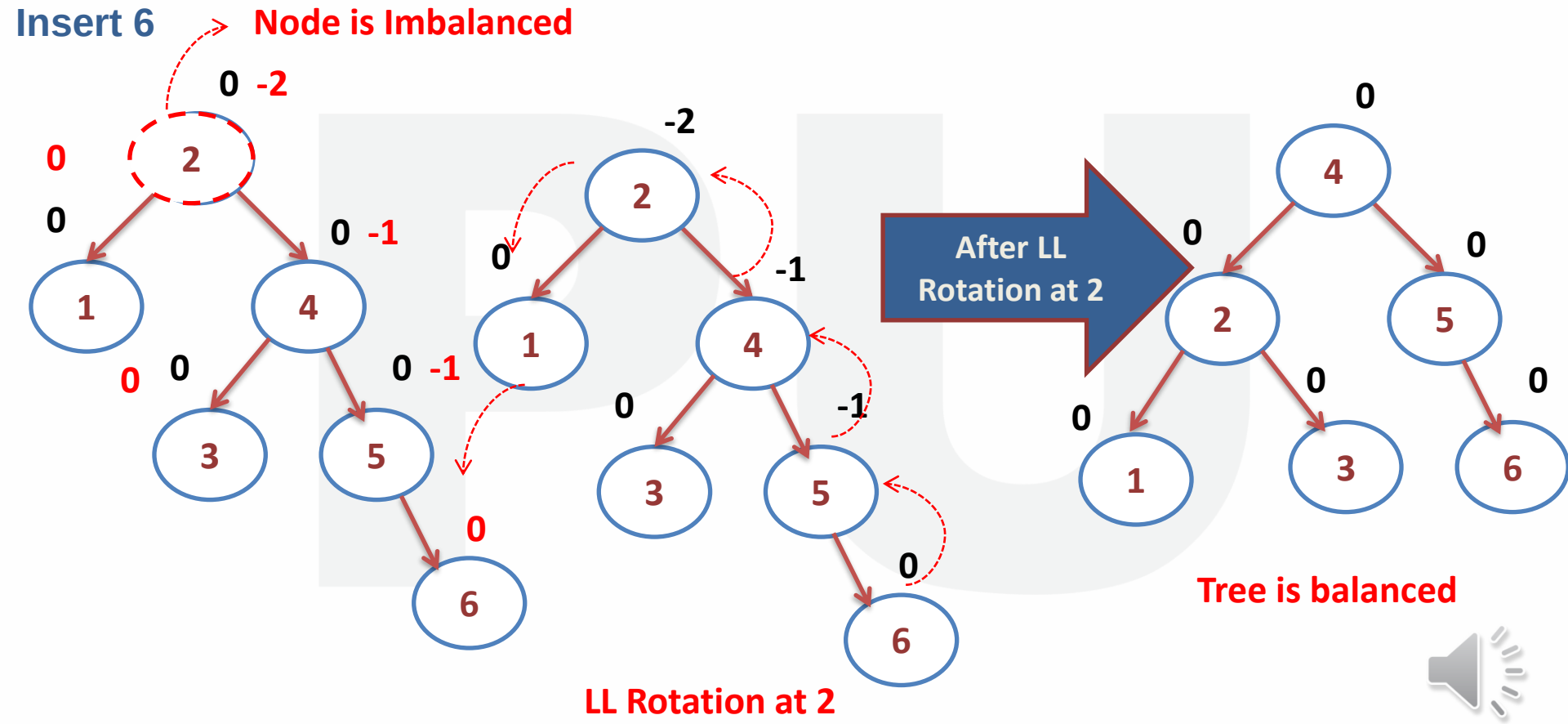
Tree is balanced

Construct an AVL tree by inserting numbers from 1 to 7

Insert 5

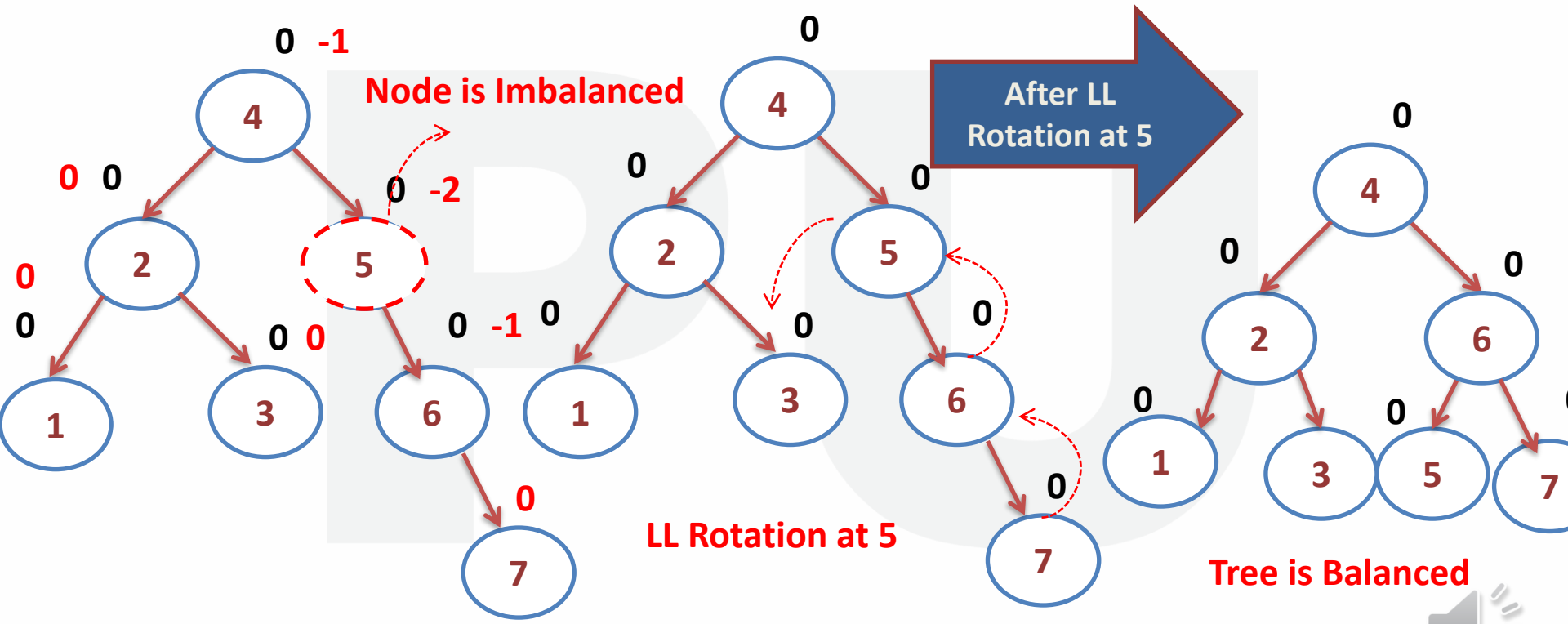


Construct an AVL tree by inserting numbers from 1 to 7



Construct an AVL tree by inserting numbers from 1 to 7

Insert 7



Deletion Operation in AVL Tree

- The deletion operation in AVL Tree is similar to deletion operation in BST.
- But after every deletion operation, we need to check with the Balance Factor condition.
- If the tree is balanced after deletion go for next operation otherwise perform suitable rotation to make the tree Balanced.

Parul® University

UNIT 4
Tree

Prof. Shyambabu PAndey, Assistant Professor
Computer Science & Engineering

Parul®
University

DIGITAL LEARNING CONTENT

Topic3

B & B+ Tree

M-Way Search Tree

Issue with Binary Search Tree (BST): Binary search tree is a good idea, but it stores only one value at each node. Suppose we have a large number of values or we want to store more than one value at one node then Binary search tree is not applicable. So, we have to go for Multiple Way Search Tree. It is also known as M -Way Search Tree or M-ary tree.

Image source : Google

M-Way Search Tree

M-way search tree is a tree that can have more than two children.

An m-way is tree of order m (m children) in which:

1. Each internal node (except root) can have at most m children/nod and each node can contain at most m-1 value.

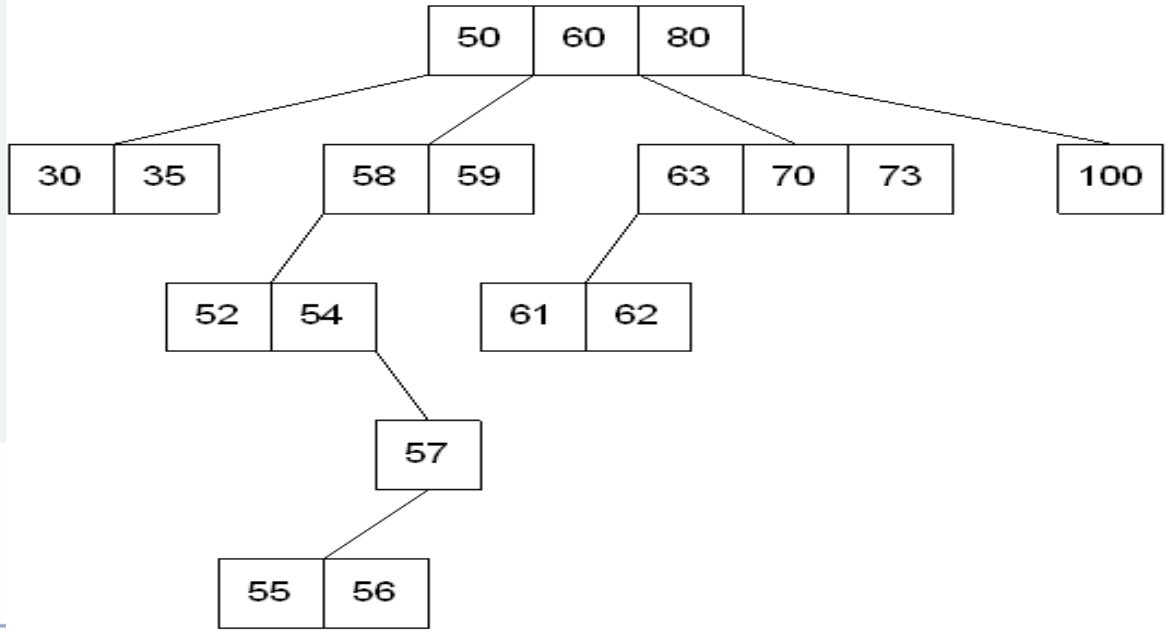
2. The values in each node are in ascending order.

3. The values in the first i children are smaller than the i-th value.

4. The values in the last m-i children are larger than the i-th value.

M-Way Search Tree

Example: Let us take a 4-way search tree.
Here the order of tree is 4 so the value of m is 4. We can see each node has at most 4 children and each node can contain at most 3 (m-1) nodes.



M-Way Search Tree

Issue with M-Way Search Tree: M-way search tree is not a height balanced tree. It will take more time for insertion and deletion operation of a node.

B-Tree

B Tree was developed by **Bayer and McCreight** in 1972 with the name **Height Balanced m-way Search Tree**. Later it was named as B-Tree.

A B-Tree is a height balanced m-Way search tree, In which:

- If the degree of tree is m then each node (Internal node except root node) can contain at most m children/nodes and at least [m/2].

Image source : Google

B-Tree

- If the degree of the tree is m, then each node must have at least [m/2]-1 keys and maximum of m-1 keys.
- Root nodes can have a minimum of two children.
- All the leaf nodes should have the same depth.

Image source : Google

B-Tree

Operations on B-Tree:

- Searching
- Insertion
- Deletion

Image source : Google

B-Tree

Searching in B-Tree: Suppose we want to search an element ‘X’ In a B-Tree, then the following will be steps:

1. First of all compare ‘X’ with the first element of the root node in the tree.
2. If comparison is matched, then display "Given element is found" and terminate the function.
3. If comparison is not matched, then check ‘X’ is smaller or larger than that key value (That means first element of root node).

Image source : Google

B-Tree

4. If ‘X’ is smaller, then go to the left subtree and continue the search process.

5. If ‘X’ is larger, then compare ‘X’ with the next key value in the same node and repeat steps 2, 3, 4 and 5 until we find the exact match or until the ‘X’ is compared with the last key value in the leaf node.

6. If the last key value in the leaf node is also not matched then display "Element is not found" and terminate the function.

B-Tree

Construct a B-Tree of order 3 by using following key values 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

Here order of B-Tree is 4. So m = 4

- No. of at most children of internal node except root node = m = 4
- No. of at least children node of internal node except root node m/2 = 2
- No. of at most key value at every node of tree= m-1 = 3
- No. of at least key value at every node of tree= m/2-1 = 1

B-Tree

Insert 10

Insert 20

Image source : Google

B-Tree

Insert 30

10	20	30

B-Tree

Insert 40

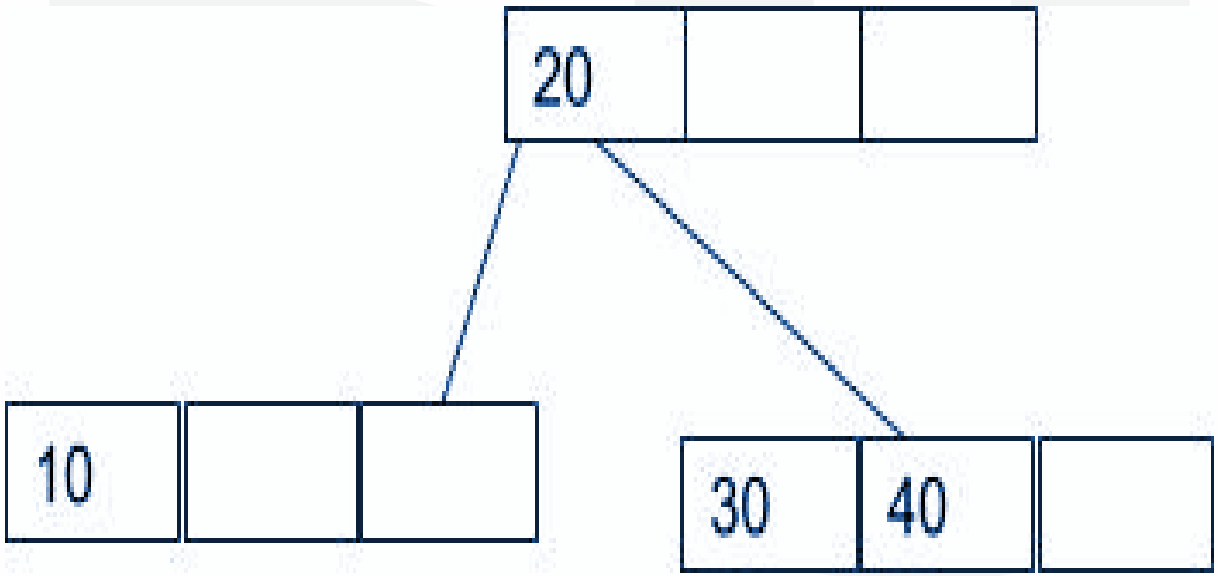
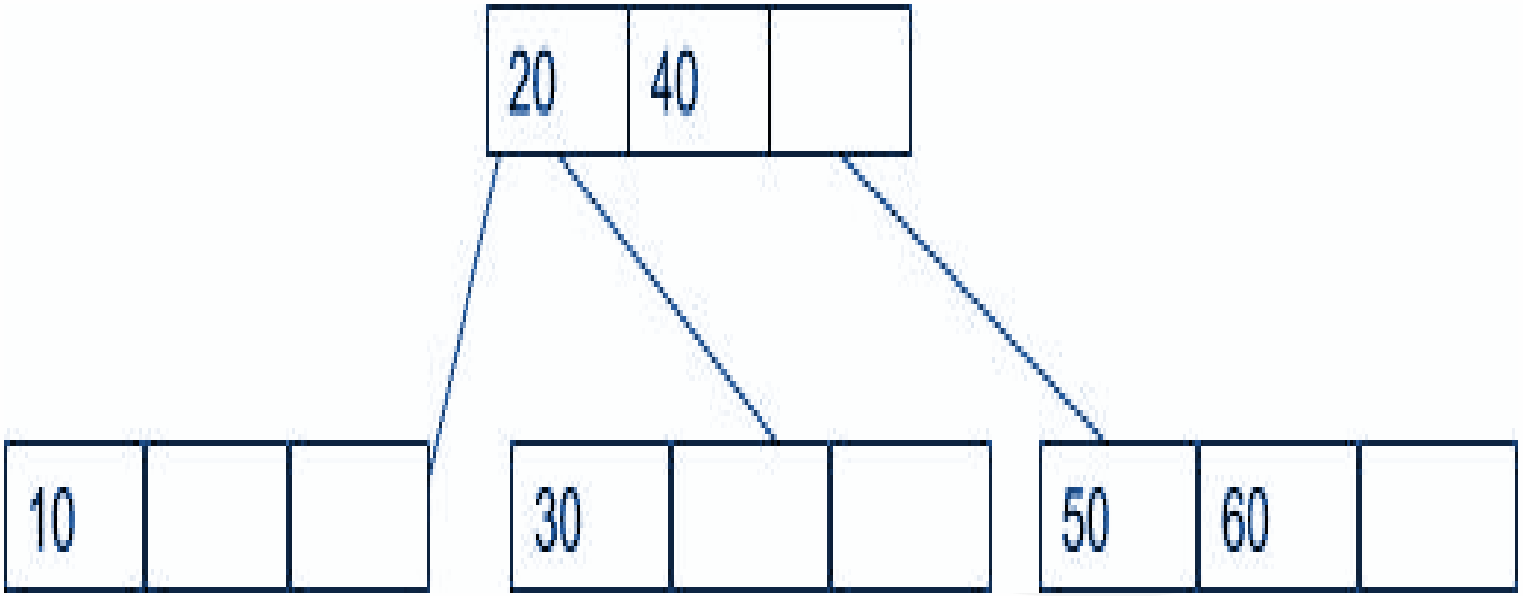


Image source : Google

B-Tree

Insert 60



B-Tree

Insert 70

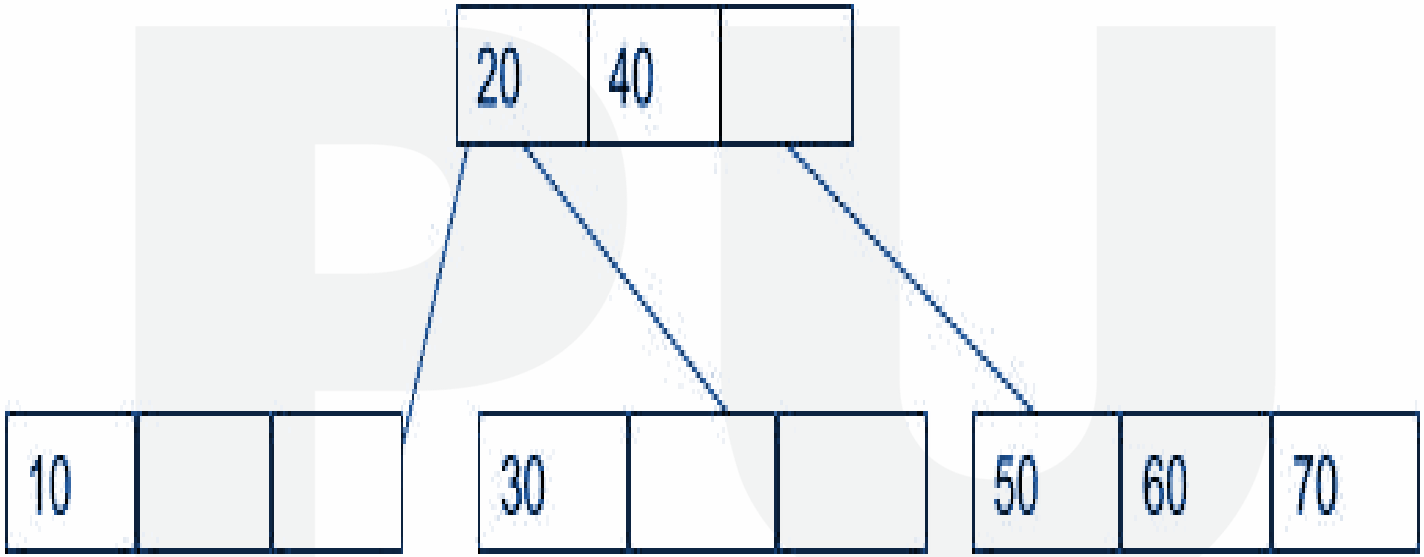


Image source : Google

B-Tree

Insert 80

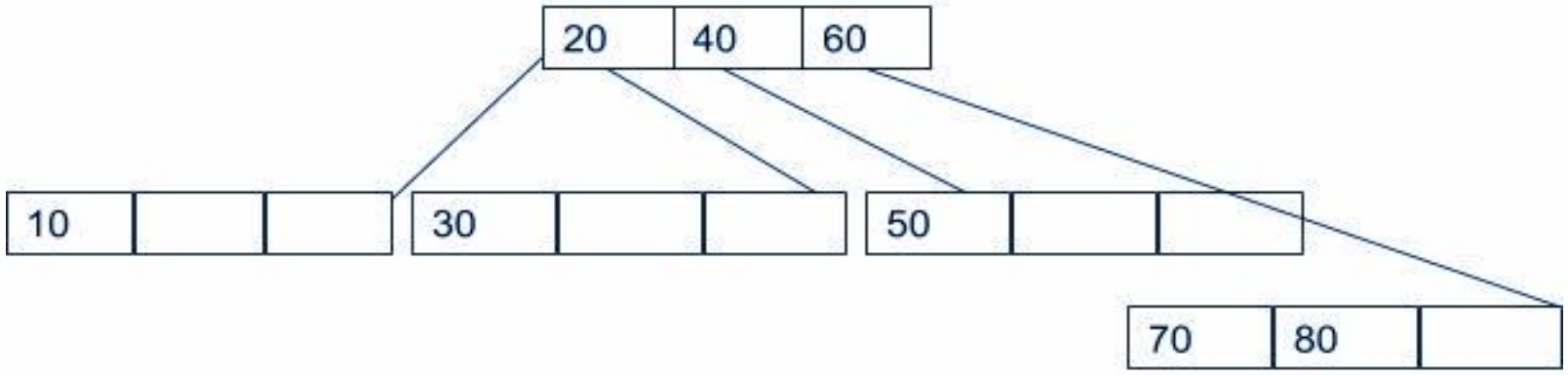


Image source : Google

B-Tree

Insert 90

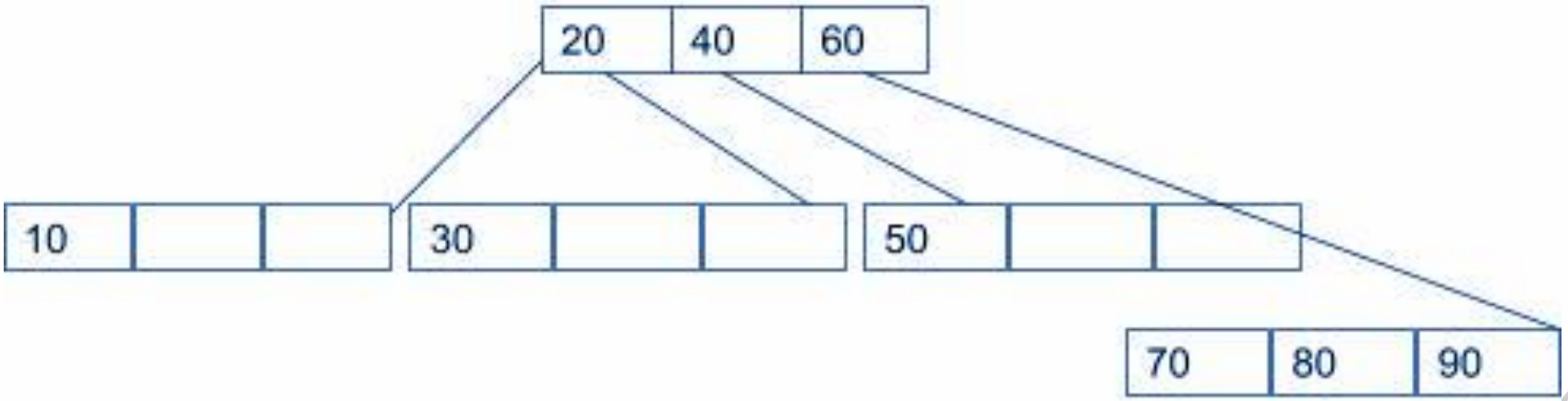


Image source : Google

B-Tree

Time Complexity:

- **Best case :** O(1) (constant time). When a key is inserted in the root node, it will take constant time for insertion.

- **Average case** : O(logn) (logarithm time). When a key is inserted in the internal node then, it will take logarithm time for insertion.

- **Worst case** : O(logn) (logarithm time). When a key will be inserted in the lea node, it will take logarithm time for insertion.

Image source : Google

B-Tree

Deletion in B-Tree:

- Apply the search operation in B-Tree and find the target key in the nodes.
- After finding the location of the target key, There are two conditions based on the location of the target key.

a. **If the** target **key is in the leaf node:**

1. If the target key is in the leaf node and the number of keys are more than min keys in the node. Deleting the target key does not violate any rule of B-Tree, so directly delete the target key.

B-Tree

2. If the target is in leaf node and it has min key nodes, then Deleting of target key will violate the property of B Tree.

- Target node can borrow key from immediate left node, or immediate right node (sibling)
- The sibling will say yes if it has more than minimum number of keys
- The key will be borrowed from the left sibling node, the max value will be transferred to the target node. Or, The key will be borrowed from the right sibling node, the minimum value will be transferred to the target node.

Image source : Google

B-Tree

3. Target is in the leaf node, but no siblings have more than min number of keys

- If both the immediate sibling nodes already have a minimum number of keys, then merge the node with either the left sibling node or the right sibling node. This merging is done through the parent node.

b. If the target key is in an internal node:

- Either choose, in- order predecessor or in-order successor

- In case the of in-order predecessor, the maximum key from its left subtree will be selected

Image source : Google

B-Tree

- In case of in-order successor, the minimum key from its right subtree will be selected
- If the target key's in-order predecessor has more than the min keys, only then it can replace the target key with the max of the in-order predecessor
- If the target key's in-order predecessor does not have more than min keys, look for the in-order successor's minimum key.
- If the target key's in-order predecessor and successor both have less than min keys, then merge the predecessor and successor.
- But, if the sibling also has only a minimum number of keys then, merge the node with the sibling along with the parent. Arrange the children accordingly (increasing order).

B-Tree

Example: Let us take a B tree of order 4
Here m=4
minimum number key of each node except root node: m/2-1= 1

Image source : Google

B-Tree

Delete 90

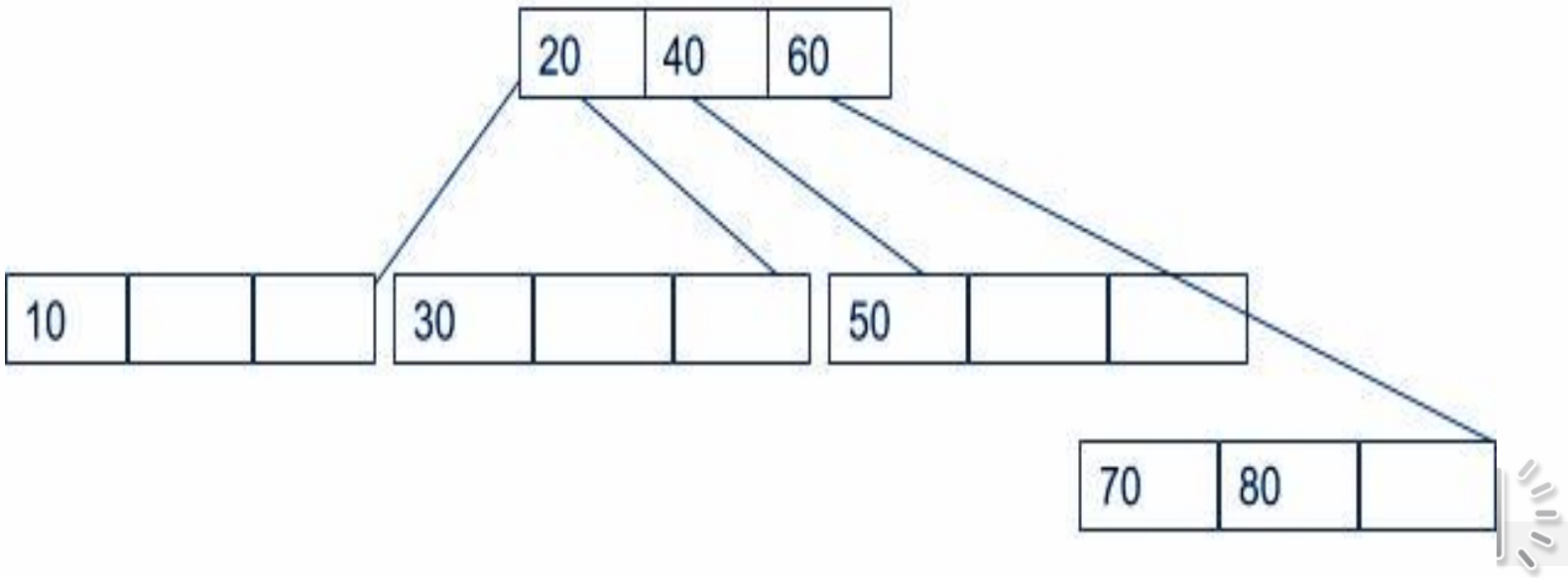


Image source : Google

B-Tree

Delete 50

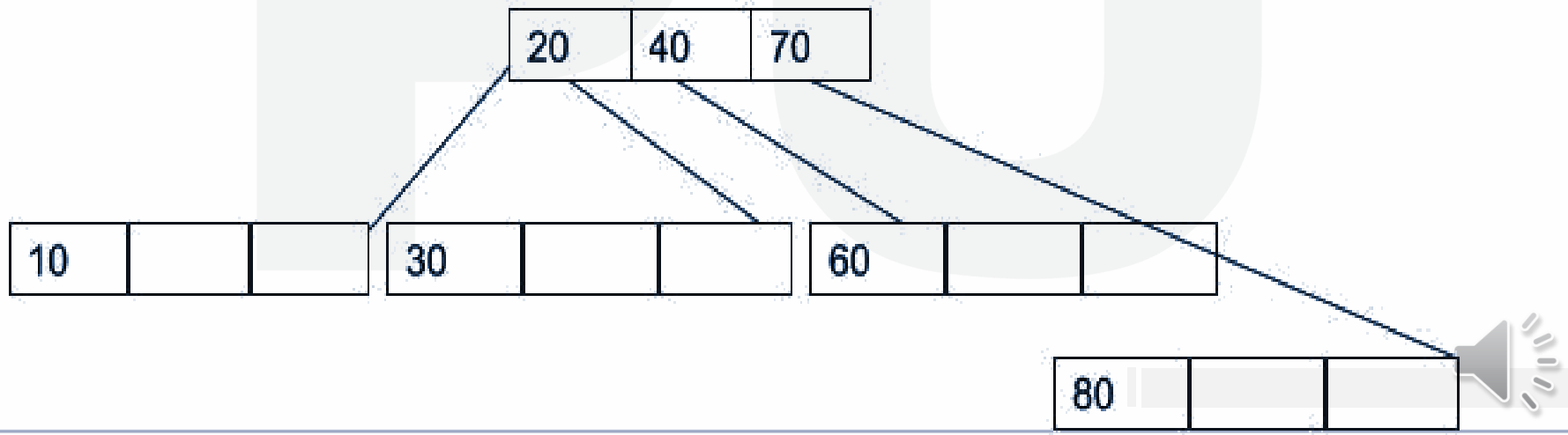
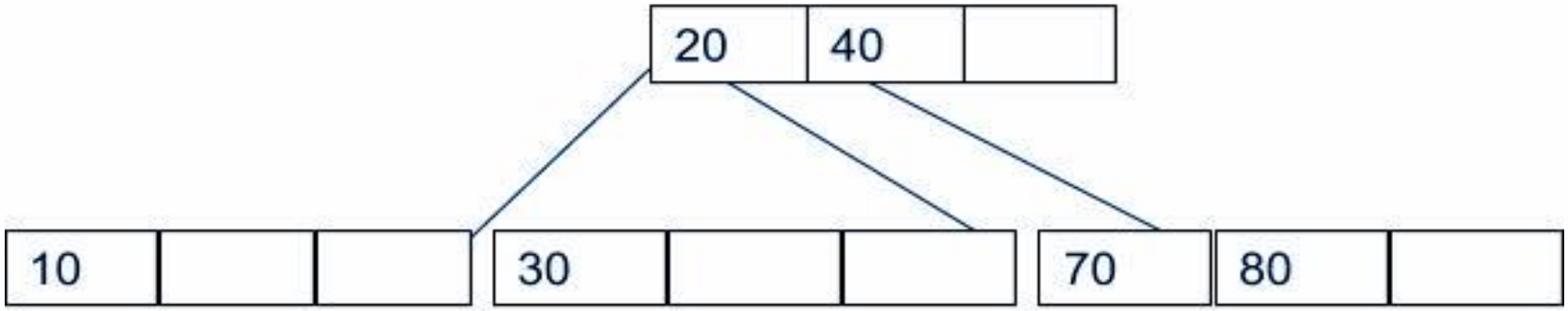


Image source : Google

B-Tree

Delete 60



B-Tree

Time Complexity:

- **Best case :** O(1) (constant time). When a key is present in the root node, it will take constant time for deleting.

- **Average case** : O(logn) (logarithm time). When a key is in the internal node then, it will take logarithm time for deleting.

- **Worst case** : O(logn) (logarithm time). When a key will be inserted in the leaf node, it will take logarithm time for deleting.

B+ - Tree

•B+ Tree is an extension of B Tree which allows efficient insertion, deletion and search operations.

B+-Tree is a height balanced M-Way search tree, In which:

•If the degree of tree is M then each node(Internal node) can contain at most m children/nodes and at least [M/2].
•If the degree of the tree is M, then each node can have at least [m/2]-1 keys and maximum of m-1 keys.
•Root nodes can have a minimum of two children.
•All the leaf nodes should have the same depth.
•Values are stored only on leaf nodes.
•The leaf nodes of a B+ tree are linked together in the form of singly linked lists to make the search queries more efficient.

Parul®
University

DIGITAL LEARNING CONTENT

Operation on B+ - Tree

- Insertion
- Deletion
- Searching

Searching in B-Tree

The search operation in B+-Tree is similar to the search operation in B Search Tree. Suppose we want to search ‘X’ In a B-Tree, then the following will be steps:

1. First of all compare ‘X’ with the first element of the root node in the tree.
2. If comparison is matched, then display "Given element is found" and terminate the function.
3. If comparison is not matched, then check ‘X’ is smaller or larger than that key value (That means first element of root node).
4. If ‘X’ is smaller, then go to the left subtree and continue the search process.

Searching in B-Tree

5. If ‘X’ is larger, then compare ‘X’ with the next key value in the same node and repeat steps **2, 3, 4 and 5** until we find the exact match or until the ‘X’ is compared with the last key value in the leaf node.

6. If the last key value in the leaf node is also not matched then display "Element is not found" and terminate the function.

Time Complexity

- **Best case :** O(1) (constant time). When a key is present in the root node, it will take constant time for searching.
- **Average case :** O(logn) (logarithm time). When a key is in the internal node then, it will take logarithm time for searching
- **Worse Case :** O(logn) (logarithm time). When a key will be inserted in the leaf node, it will take logarithm time for searching.

Insertion in B-Tree

Always, New elements will be inserted at leaf nodes in B+-Tree. Suppose we want to insert a key value ‘X’, then the following will be steps to insert a new element in B+-Tree.

1. First of all check whether the tree is empty.

2. If the tree is empty, then create a node with ‘X’ and insert it in the tree as a root node.

3. If the tree is not empty, then insert ‘X’ into the suitable leaf node by using Binary Search Tree logic.

Insertion in B-Tree

4. If the respective leaf node has some empty position, then insert ‘X’ in that node in ascending order of key value.
5. If the respective leaf node is full, then split that node and send a copy of the middle value to its parent. Repeat the same procedure until ‘X’ will be fixed into a node.
6. If the splitting is performed at root node then the middle value becomes a new root node for the tree and the height of the tree is increased by one.

Insertion in B-Tree

Example: Construct a B+-Tree of order 4 for the given elements 10, 20, 30, 40, 50, 60, 70, 80, 90.

Here m=4

Maximum number of children nodes of internal nodes: 4

Minimum number of children nodes of internal nodes: m/2= 2

Maximum number of keys on each node: m-1= 3

Minimum number of keys on each node: m/2-1= 1

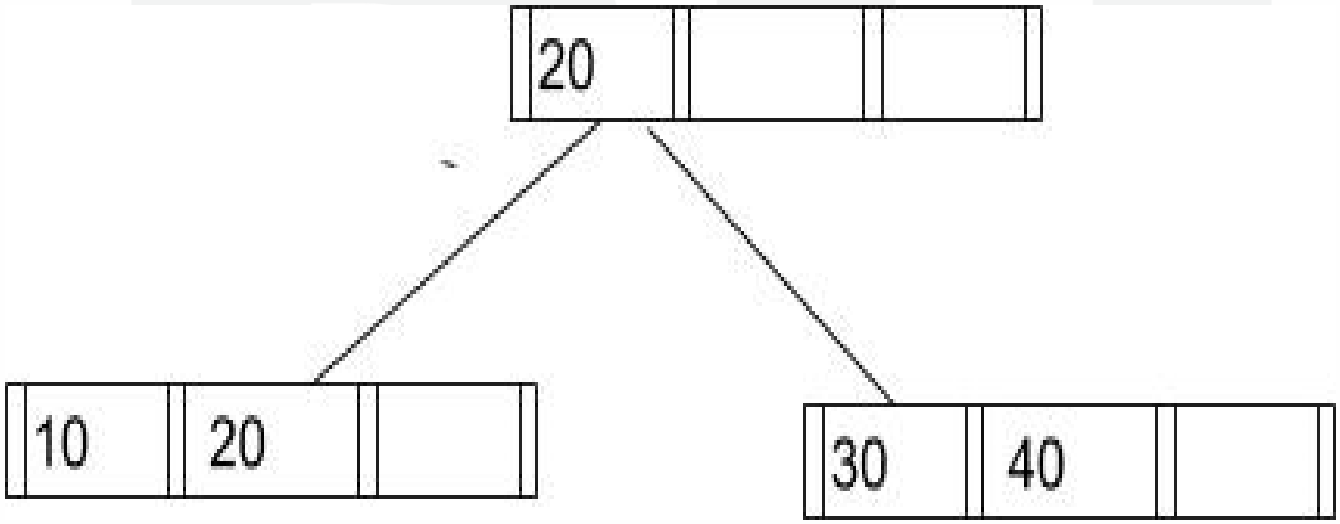
Insertion in B-Tree

Insert 10

Insert 20, 30

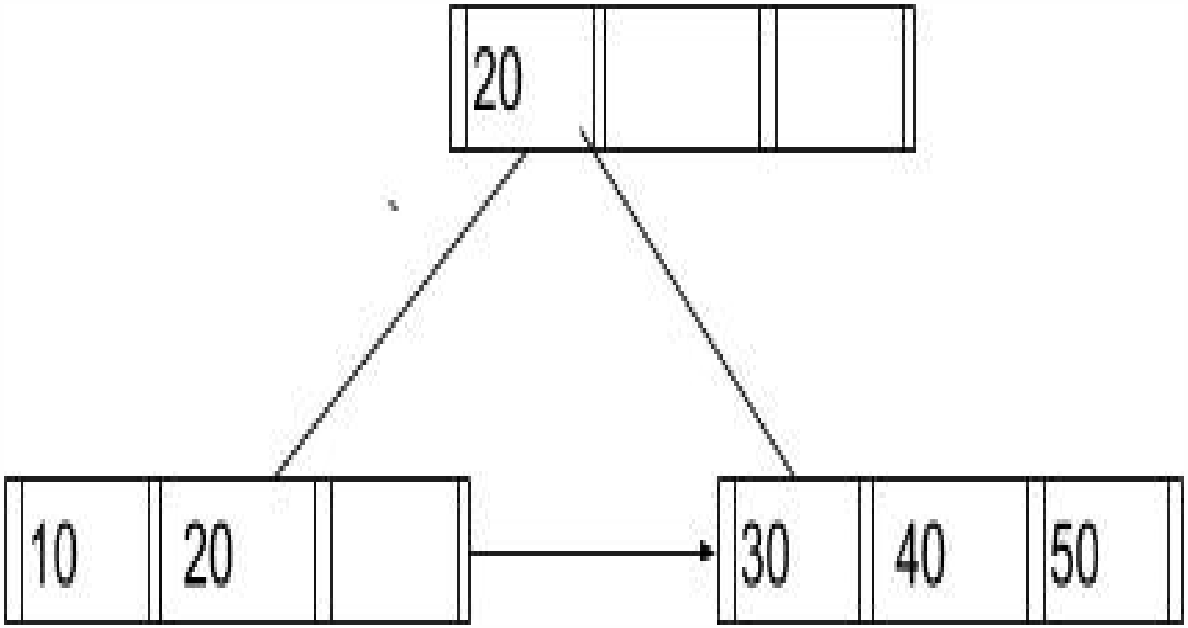
Insertion in B-Tree

Insert 40



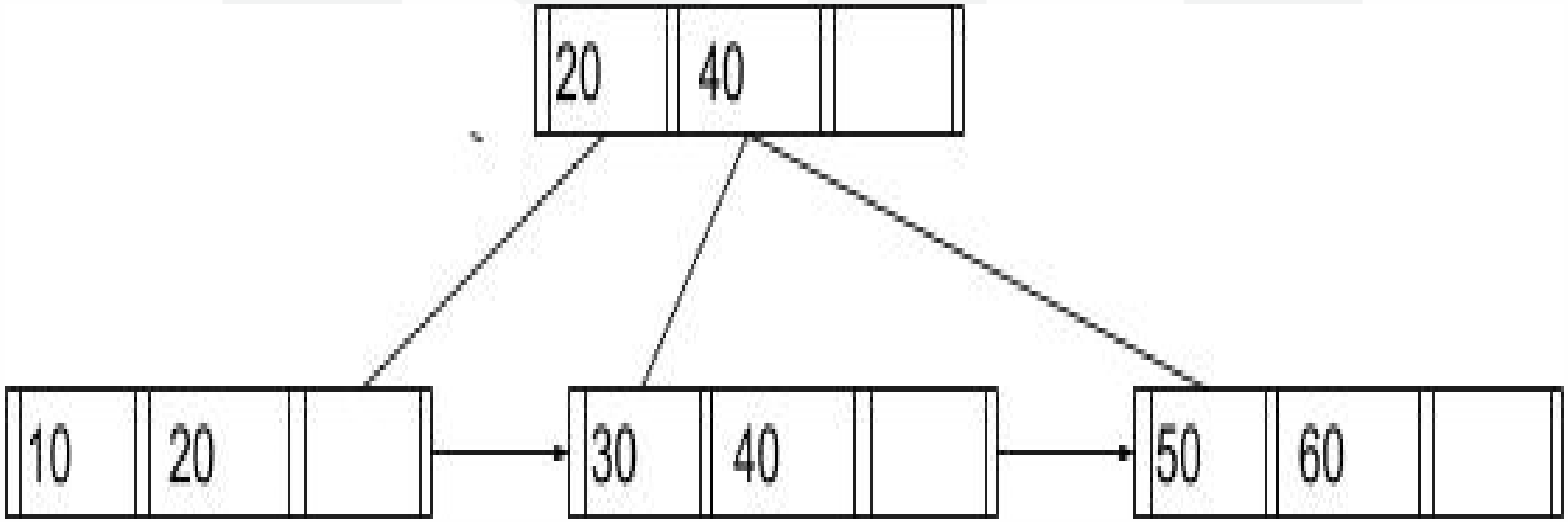
Insertion in B-Tree

Insert 50



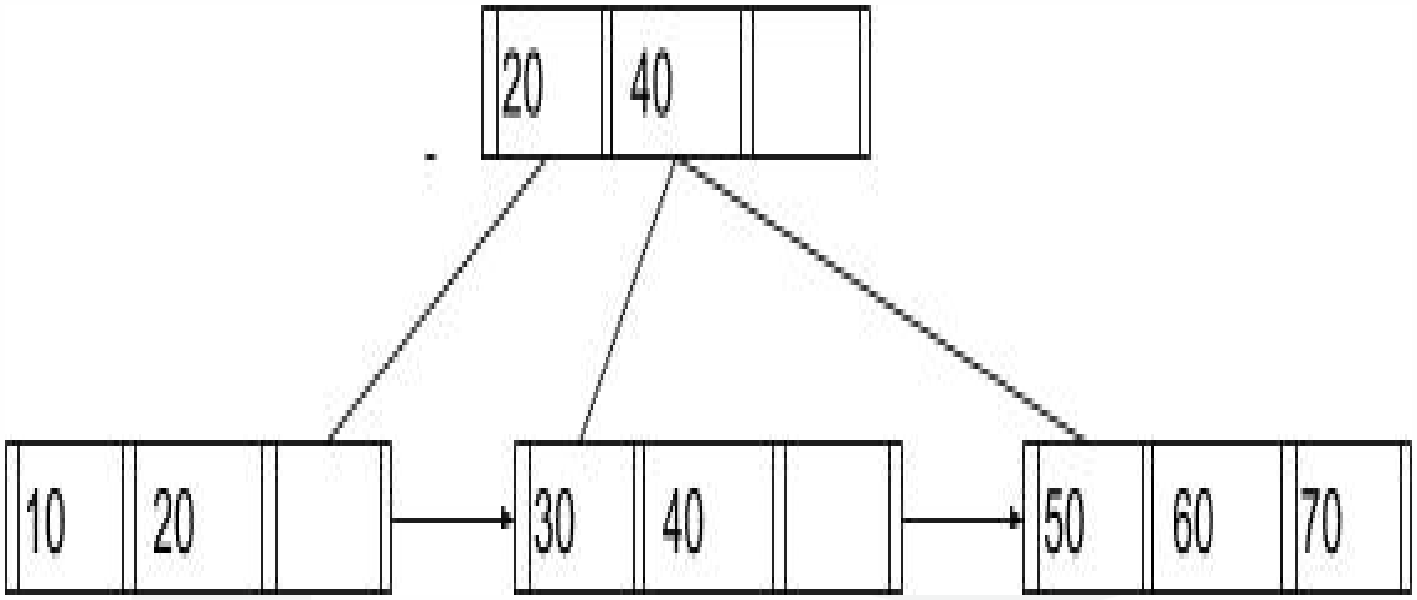
Insertion in B-Tree

Insert 60



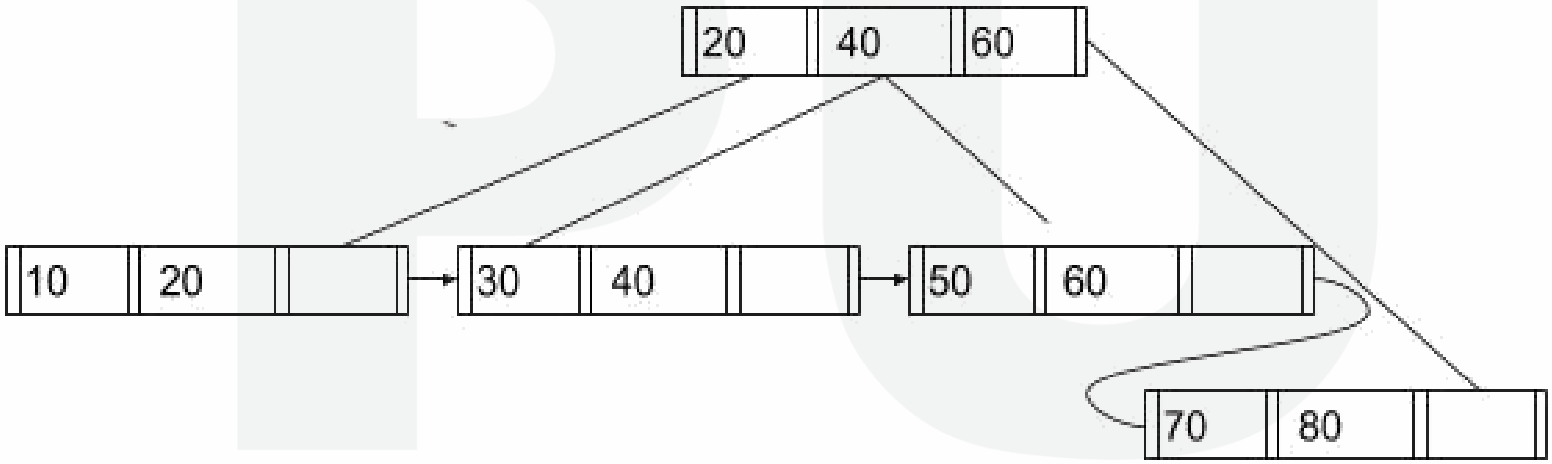
Insertion in B-Tree

Insert 70



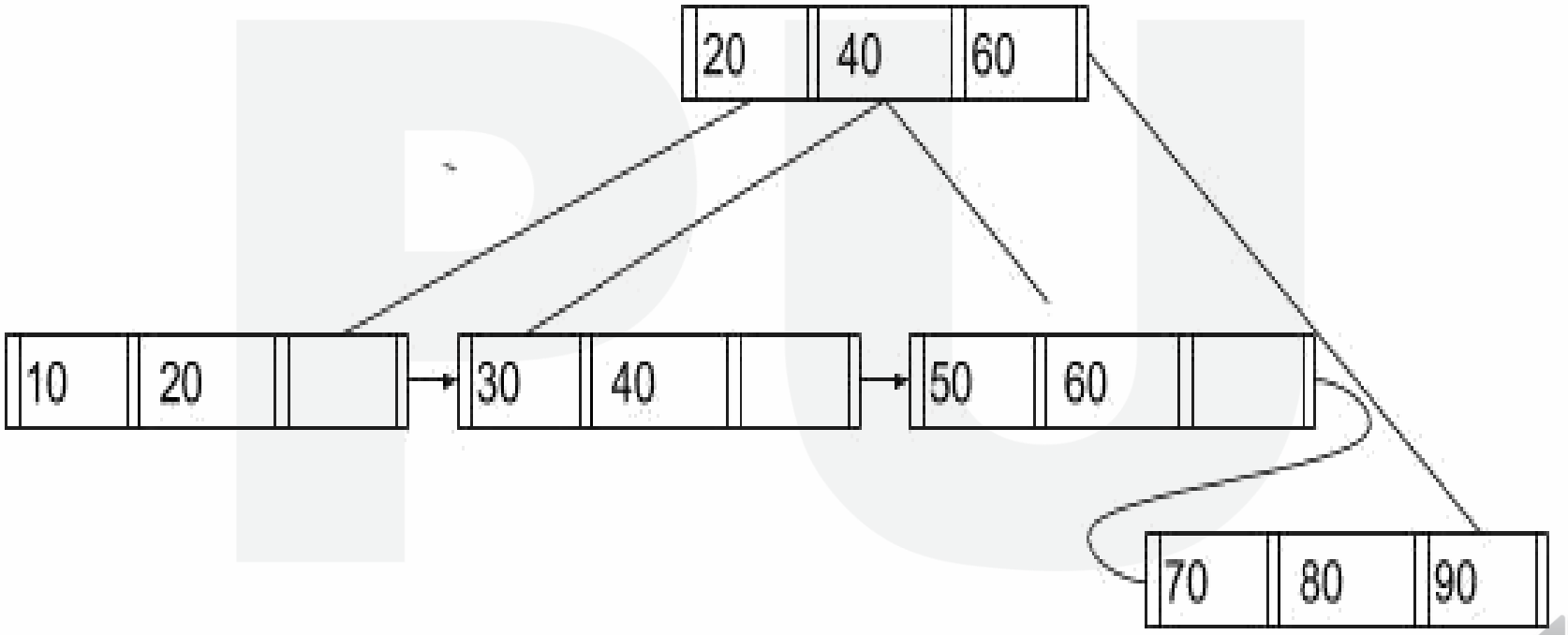
Insertion in B-Tree

Insert 80



Insertion in B-Tree

Insert 90



Deletion in B-Tree

For Deletion of a node from B+-Tree, we have to apply the following procedure:

1. Apply the search operation in B+-Tree and find the target key in the nodes.
2. After finding the position of the target key in leaf nodes, There are two conditions based on the location of the target key in leaf node.

a. The target key is present only at the leaf node not in the indexes (or internal nodes). There are two cases for it:

case 1: There is more than the minimum number of keys in the node. Simply delete the key

case 2: There is an exact minimum number of keys in the node. Delete the key and borrow a key from the immediate sibling. Add the median key of the sibling node to the parent.

Deletion in B-Tree

b. The key to be deleted is present in the internal nodes as well. Then we have to remove them from the internal nodes as well. There are the following cases for this situation.

Case 1: If there is more than the minimum number of keys in the node, simply delete the key from the leaf node and delete the key from the internal node as well. Fill the empty space in the internal node with the inorder successor.

Case 2: There is an exact minimum number of keys in the node. Delete the key and borrow a key from the immediate sibling. Add the median key of the sibling node to the parent.

Case 3: This case is similar to Case 1 but here, empty space is generated above the immediate parent node. After deleting the key, merge the empty space with its sibling. Fill the empty space in the grandparent node with the inorder successor.

Deletion in B-Tree

Example: Let take a B+-Tree order of 4.

Here m=4

Maximum number of children nodes of internal nodes: 4

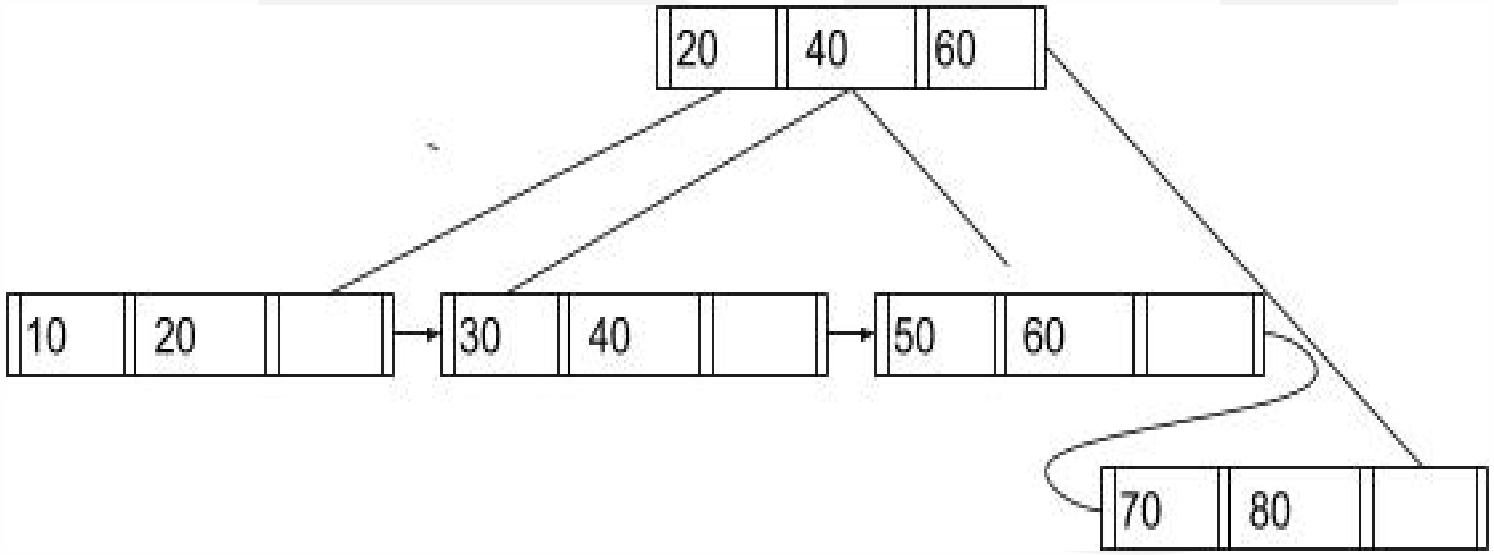
Minimum number of children nodes of internal nodes: m/2= 2

Maximum number of keys on each node: m-1= 3

Minimum number of keys on each node: m/2-1= 1

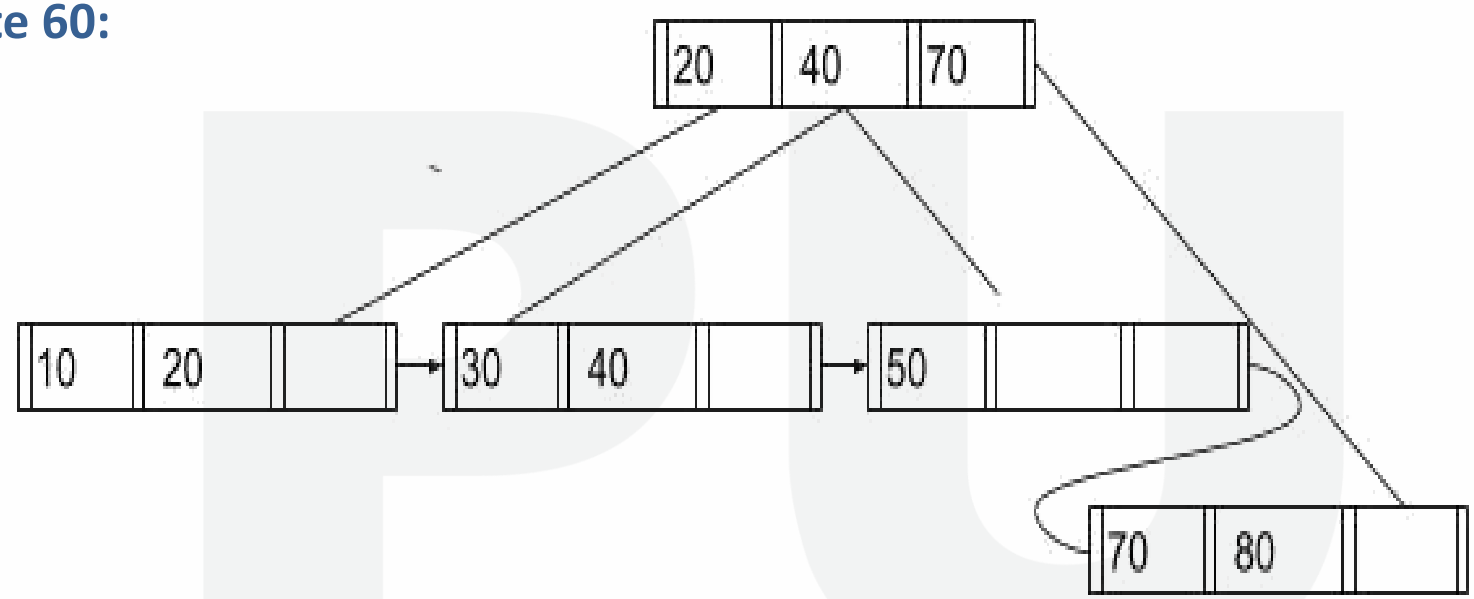
Deletion in B-Tree

Delete 90:



Deletion in B-Tree

Delete 60:



Differenence between B-Tree & B+-Tree

B-Tree	B+-Tree
Data is stored in leaf nodes as well as internal nodes.	Data is stored in leaf nodes as well as internal nodes.
Searching is a bit slower as data is stored in internal as well as leaf nodes.	Searching is faster as the data is stored only in the leaf nodes.
No redundant search keys are present.	No redundant search keys are present.
Deletion operation is complex.	Deletion operation is easy as data can be directly deleted from the leaf nodes.
Leaf nodes cannot be linked together.	Leaf nodes are linked together to form a linked list.

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in