S4 DATASTRUCTURES(DS) Code: 4133

MODULE 3 - TREES

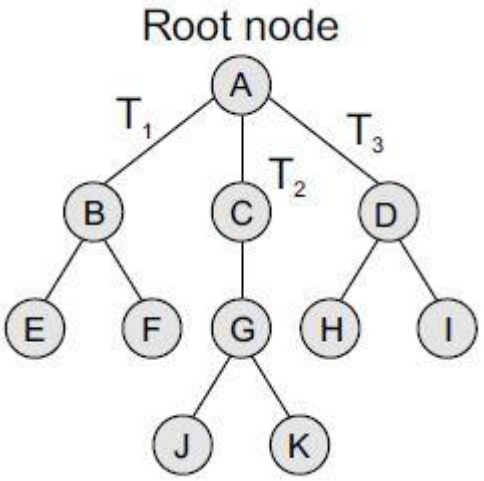
SYLLABUS

3.1 Understanding Trees and its operations

3.1.1 Explain binary tree, key terms related to trees and traversal methods.
3.1.2 Explain Linked representation of binary trees
3.1.3 Explain binary search trees (BST) and its operations – traversals, insertion, deletion andfind.
3.1.4 Describe BST ADT with inOrder(), preOrder(), postOrder(), insert(), delete(), find() etc.
3.1.5 Describe Expression trees and Threaded binary trees.

Binary trees

- A tree is a non-linear data structure which consists of a collection of nodes arranged in a hierarchical order.
- Every tree has a root element pointed by a 'root pointer'.
- A tree is defined as a set of one or more nodes where one node is represented as root of the tree and all the remaining nodes can be partitioned into non-empty sets each of which is a sub-tree of the root.



Basic Terms

Root node

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

Leaf node

- A node that has no children is called the leaf node or the terminal node.
- From the figure E,F,J,K,H and I are leaf nodes.

Sub-trees

- If the root node R is not NULL, then the trees T1,T2,T3 are called the sub trees of R

Path

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

Ancestor node

- An ancestor of a node is any previous node(predecessor) on the **path from root to that node.**
- The root node does not have any ancestors.
- In figure the ancestors of nodes k is A,C,G

Descendant node

- A descendant node is any successor node on any **path from the node to a leaf node**.
- Leaf node do not have any descendants.
- In figure : the descendants of node C are node G,J,K:the descendants of node A are nodes C,G,J,K

Level Number

- Every node in the tree is assigned a level number in such a way that the root node is at level 0, children of the root node are at level number 1.
- Thus every node is at one level higher than its parent.so, all child nodes have a level number given by **parent's level number +1**.

Degree

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

In-degree

- In-degree of a node is the **number of edges arriving at that node**.

Out-degree

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

TYPES OF TREES

There are 7 types of trees

1) General trees
2) Forests
3) Binary trees
4) Binary search trees
5) Expression trees
6) Tournament trees
7) Threaded binary trees

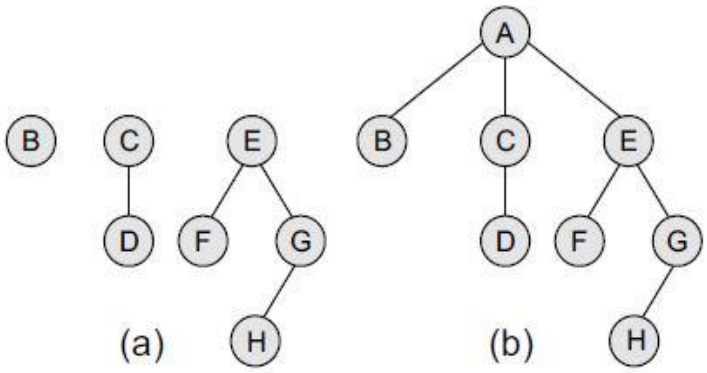
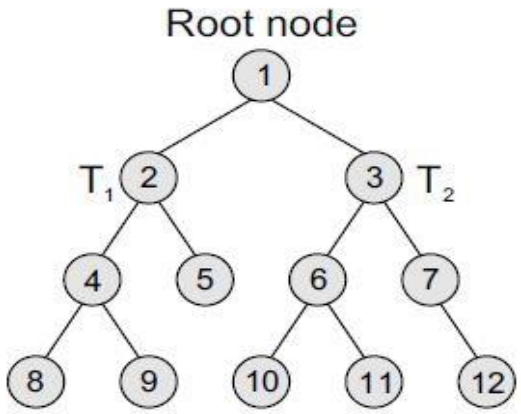


Fig. Forest and its corresponding tree

BINARY TREES

- **A binary tree is a data structure that is defined as a collection of elements called nodes**.
- In a binary tree the **top most element** is called the **root node**, **and each node has 0,1 or at the most two children.**
- A node that has zero children is called a leaf node or a terminal node.
- Every node obtains a data element a left pointer which points to the left child, and a right pointer which points to the right child
- The root element is pointed by a "root" pointer. if root=NULL then the tree is empty

For example:



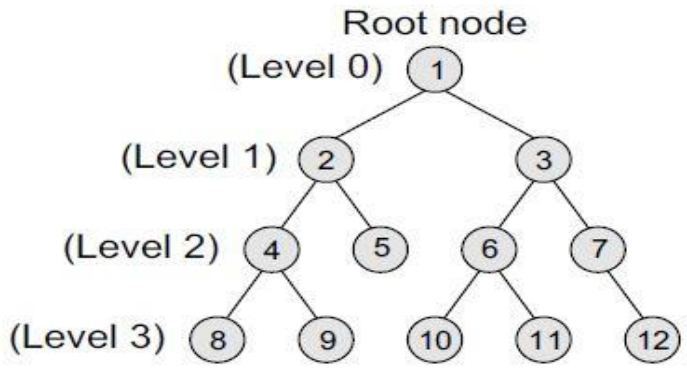
- R is the root node
- T1 is the left sub trees of R and T2 is the right sub tree of R
- T1 is the left successor of R.T2 is the right successor of R
- The left sub tree of the root node consists of the node: 2,4,5,8,9
- The right sub trees of the root node consists of nodes: 3,6,7,10,11,12
- In the tree, root node 1 has two successor children: 2 and 3
- Node 2 has two successor nodes, children 4 and 5
- Node 4 has two successor nodes, children 8 and 9
- Node 5,8,9,10,11,12 has no successor. It's the leaf node.
- Node 3 has two successor nodes: 6 and 7
- Node 6 has two successors: 10 and 11
- Finally node 7 has only one successor 12
- Every node in a binary tree contains a left sub-tree and a right sub tree.
- Even the leaf nodes(terminal node) contain an empty left sub-tree and an empty right subtree
- Nodes 5,8,9,11,12 have no successors and thus said to have empty subtrees.
- Binary trees are commonly used to implement binary search tree, binary heaps,tournament trees, expression trees.

Basic Terms(key terms/terminology)

Parent

- If N is any node in tree that has left successor s1 and successor s2 then N is called the parent of s1 and s2.
- Correspondingly s1 and s2 are called the left child and the right child of N.
- Every node other than the root node has a parent.

Level number



- Every node in the binary tree is assigned a level number
- The root node is defined to be at level 0
- The left and the right child of the root node have a level number 1.
- Similarly, every node is at one level higher than its parents
- So, level number of child nodes= parents level number +1

Degree of a node

- Degree is the **number of children** that a node has.
- The degree of a leaf node is zero

Degree of a node

- The **degree of a tree** is the maximum **degree** of a node in the **tree**.

Sibling

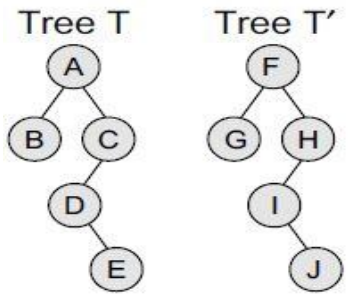
- **All nodes that are at the same level and share the same parent** are called siblings(brothers,sisters)

Leaf node

- A **node that has no children** is called a leaf node or terminal node.

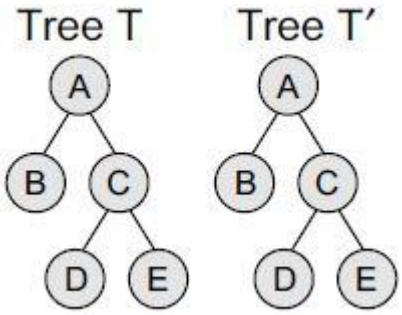
Similar binary trees

- Two binary trees T and T' are similar if **both these trees have the same structure**.



Copies

- Two binary trees T and T' are copies if they **have similar structure and if they have same content** at the corresponding nodes.



Edge

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

Path

- **A sequence of consecutive edges.**

Depth

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

Height of a tree

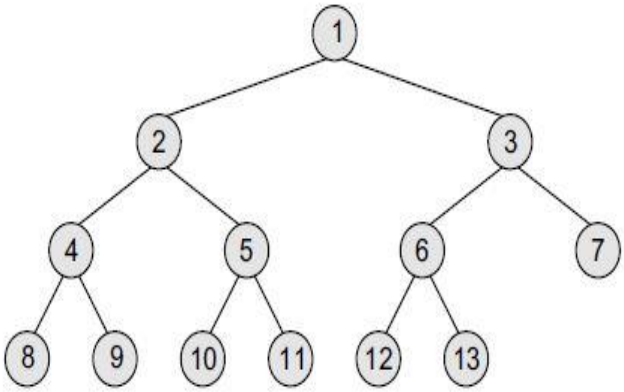
- **It is the total number of nodes on the path from the root node to the deepest node in the tree**
- A tree with only a root node has a height of 1.
- A binary tree of height h has at least h nodes and at most 2^h -1 nodes.

Complete Binary Trees

- A *complete binary tree* is a binary tree that satisfies two properties.
  - First, in a complete binarytree, every level, except possibly the last, is completely filled.
  - Second, all nodes appear as farleft as possible.

- In a complete binary tree Tn, there are exactly n nodes and level r of T can have at most 2r nodes.

- Figure shows a complete binary tree.



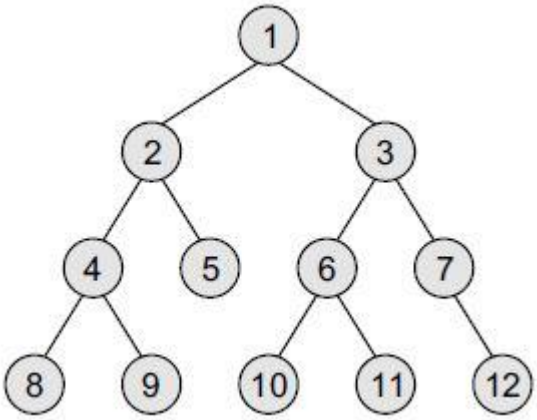
Representation of Binary Trees in the Memory

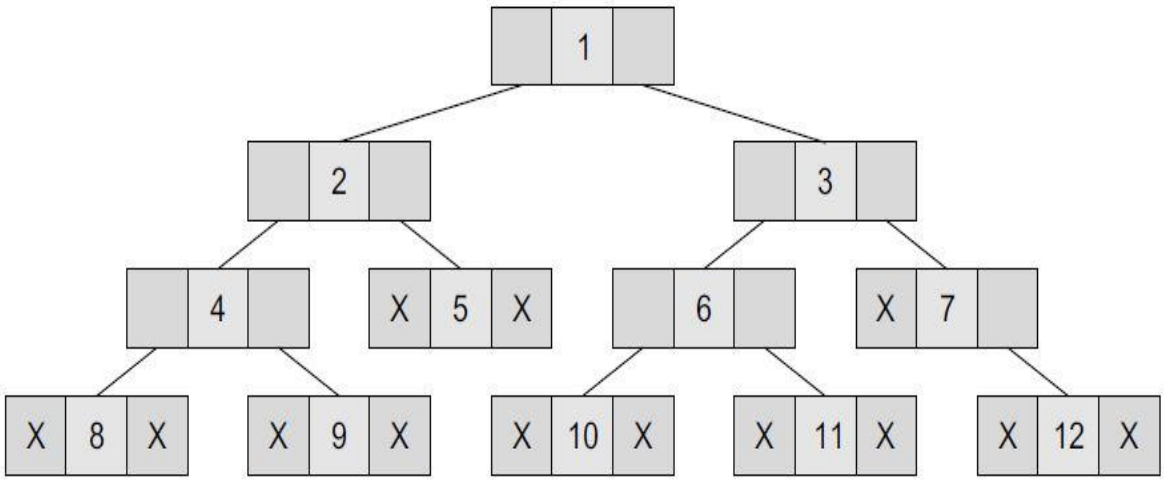
- In the computer's memory, a binary tree can be maintained either by using a linked representationor by using a sequential representation.

Linked representation of binary trees

- In the linked representation of a binary tree, every nodewill have three parts: **the data element, a pointer to the left node, and a pointer to the right node.**
- Every binary tree has a pointer ROOT, which points to the root element (topmost element) of thetree.
- If ROOT = NULL, then the tree is empty.

Consider the binary tree





- Figure shows linked representation of given tree.
- The **left position** is used to point to the left child of the node or to store the **address of the left child of the node.**
- The **middle position** is used to store the **data.**
- The **right position** is used to point to the right child of the node or to store the **address of the right child of the node.**
- Empty sub-trees are represented using X (meaning NULL).

ROOT 3

AVAIL 15

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

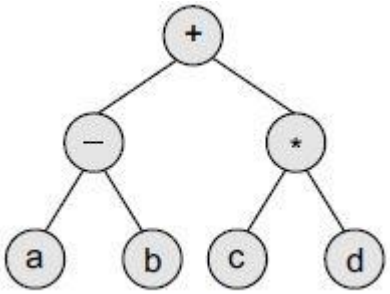
Memory representation of given tree

Expression Trees

- Expressions are represented by using tree is called Expression tree.
- Binary trees are widely used to store algebraic expressions.
- The operator is placed as corresponding parent of its operands.
  (ie, symbol as parent & character integer as child)

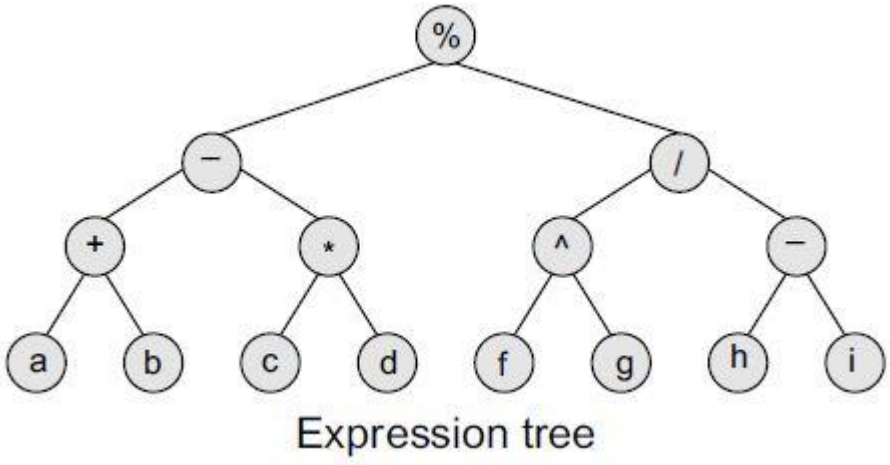
Examples

Exp = (a – b) + (c * d)



Example 2

Exp = ((a + b) – (c * d)) % ((f ^g) / (h – i))

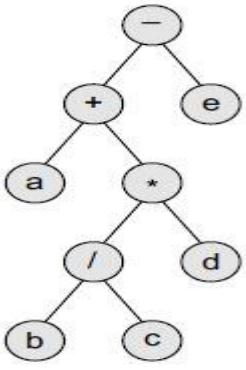


Expression tree

Example 3

Exp = a + b / c * d – e

The given expression can be written as **Exp = ((a + ((b/c) * d)) – e)**



Traversing a Binary Tree

- **This is the process of visiting each node in the tree exactly once in a systematic way.**
- Unlike linear data structure in which the elements are traversed sequentially, tree is a non-linear data structure in which the elements can be traversed in many different ways.
- There are different algorithms for tree traversal. These algorithms differ in the order in which the nodes are visited.
- **The different traversal methods are:**

1) **Pre-order traversal**
2) **Post-order traversal**
3) **In-order traversal**
4) **Level-order traversal**

Pre-order traversal(depth-first traversal)

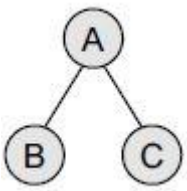
- To traverse a non-empty binary tree in pre-order the following operations are performed at each node.

1) **Visiting the root node**
2) **Traversing the left subtree**
3) **Traversing the right subtree**

Algorithm for preorder traversal

Step 1: repeat steps 2 to 4 while Tree!=Null
Step 2: write Tree→Data
Step 3: Preorder (Tree→Left)
Step 4: Preorder (Tree→Right)
Step 5: end

- Pre order algorithm is also known as the NLR traversal algorithm **(NodeRoot-Left-Right)**



Preorder traversal of given tree is **A,B,C**

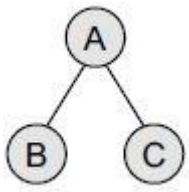
In-order traversal(symmetric traversal)

- To traverse a non-empty binary tree in in-order,the following operations are performed at each node
    1) **Taversing the left sub-tree**
    2) **Visiting the root node**
    3) **Traversing the right sub-tree**

Algorithm for in-order traversal

Step 1: repeat step 2 to 4 while Tree!=Null
Step 2 : Inorder(Tree→Left)
Step 3: write Tree→data
Step 4:Inorder(Tree→Right)
Step 5: end

- In-order algorithm is always known as the LNR traversal algorithm **(Left-NodeRoot-Right)**
- In-order traversal algorithm is usually used to display the elements of a binary search tree



Inorder traversal of given tree is **B,A,C**

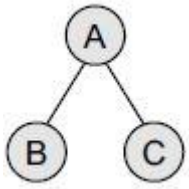
Post-order traversal

- To traverse a non-empty binary tree in post-order, the following operations are performred recursively at each node.
    1) **Traversing the left sub tree**
    2) **Traversing the right sub tree**
    3) **Visiting the root node**

Algorithm for post-order traversal

Step 1: repeat step 2 to 4 while Tree!=Null
Step 2:Postorder(Tree→Left)
Step 3: Postorder(Tree→right)
Step 4: write Tree→Data
Step 5:end

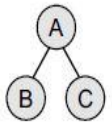
- Post-order traversal is also known as the LRN traversal algorithm(Left-Right-NodeRoot)
- Postorder traversals are used to extract postfix notation from an expression tree.



Postorder traversal of given tree is **B,C,A**

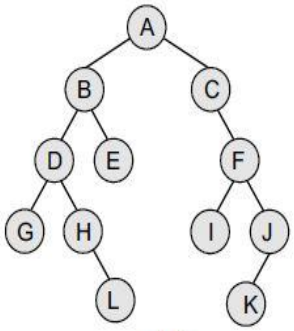
Level-order Traversal

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



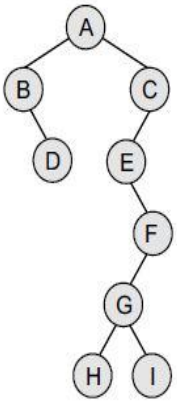
(a)

TRAVERSAL ORDER:
A, B, and C



(b)

TRAVERSAL ORDER:
A, B, C, D, E, F, G, H, I, J, L, and K



(c)

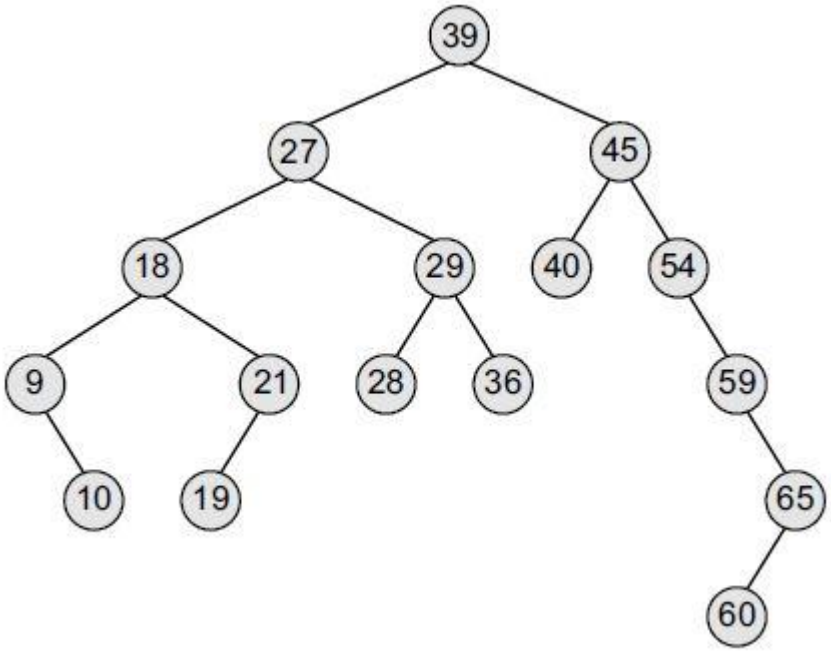
TRAVERSAL ORDER:
A, B, C, D, E, F, G, H, and I

APPLICATIONS OF TREES

1. Trees are used to store simple as well as complex data.
2. Trees are often used for implementing other types of data structures like hash tables, sets, and maps.
3. They are used to index a large number of records.
4. Trees are an important data structure used for compiler construction.
5. Trees are also used in database design.
6. Trees are used in file system directories.
7. Trees are used for information storage and retrieval in symbol tables.

Binary Search Trees

- A binary search tree is a form of binary tree in which **the nodes are arranged in an order.**
- **In a binary search tree:**
  - **All the nodes in the left sub-tree have a value less than the root node.**
  - **All the nodes in the right sub-tree have a value greater than the root node.**
  - **The same rule is applicable to every sub-tree in the tree.**
- A binary search tree also known as an **ordered binary tree**

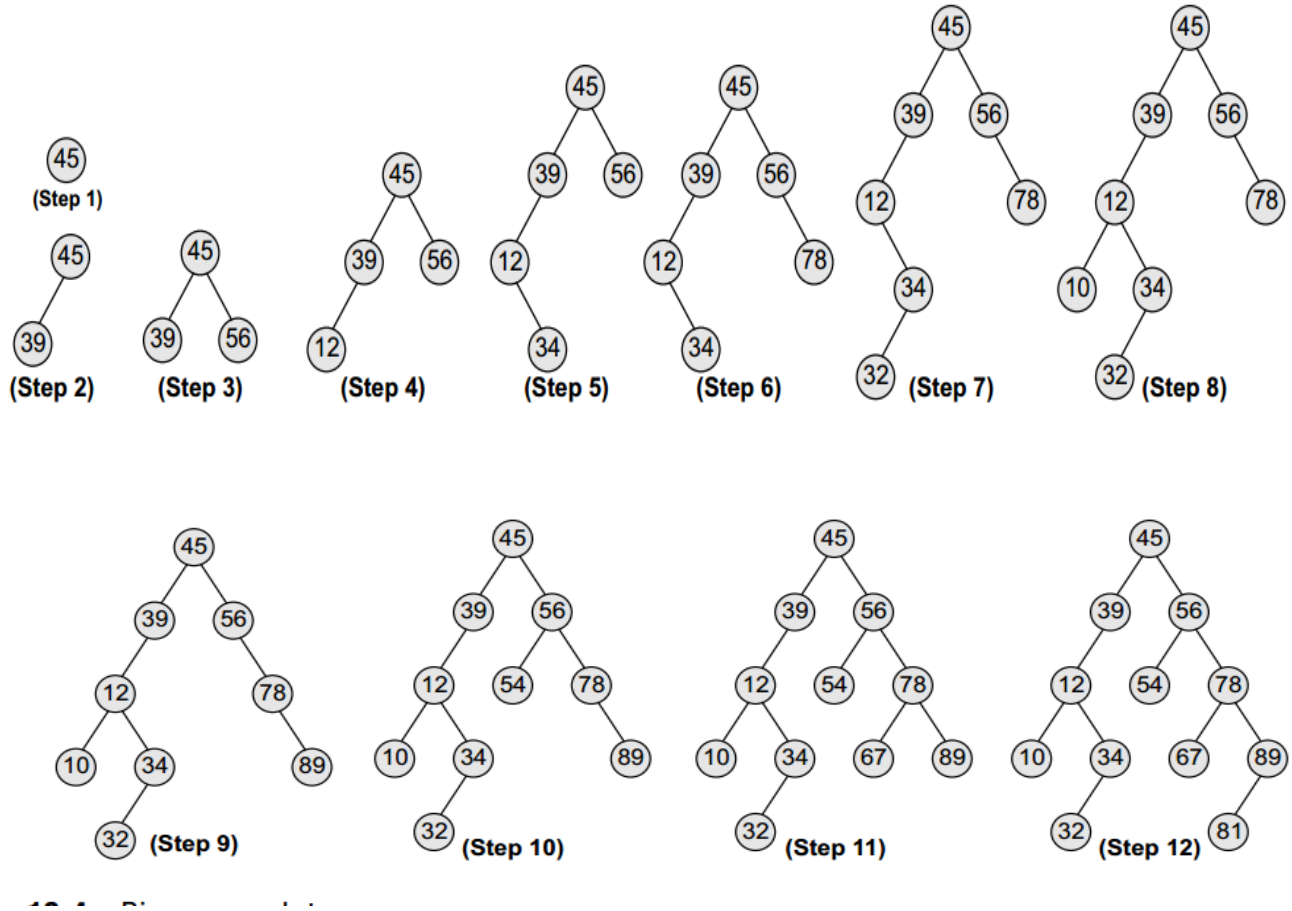


Advantages of BST

1. **The time needed to search an element** in the tree is **greatly reduced(less**). (we do not need to traverse the entire tree. At every node, we get a hint regarding which sub-tree to search in).
2. The **average running time** of search operation eliminate half of the sub-tree from the search process
3. Binary search trees also **speed up the insertion and deletion operations**. The tree has a speed advantage when the data in the structure changes rapidly.
4. **Binary search trees** are considered to be **efficient DS** when compared with stored linear array and linked list
5. **Inserting and deleting** element in a **linked list is easier** but **searching** for an element is **faster in BST.**

Example

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



Operations on Search Tree

- **Traversals**
- **Find, Search**
- **Insertion**
- **Deletion**

Searching for a node in a binary search tree

- **The search function is used to find whether a given value is present in the tree or not.**
- The function first check if the binary search tree is empty. If it is empty, then the value we are searching for is not present in the tree. So the search algorithm terminates by displaying an appropriate message.
- The searching process begins at the root node.

- If there are nodes in the tree, than the search functions checks to see if the key value of the current node is equal to the value to be searched.
- If not, it checks if the value to search is less than the current node, in this case it should move on the left child node
- In case the value is greater than the value of the current node, it should move on the right child node.

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

SearchElement (TREE, VAL)

Step 1: IF **TREE ->DATA = VAL** OR TREE = NULL
Return TREE
ELSE
IF **VAL < TREE ->DATA**
Return searchElement(TREE -> LEFT, VAL)
ELSE
Return searchElement(TREE -> RIGHT, VAL)
[END OF IF]
[END OF IF]
Step 2: END

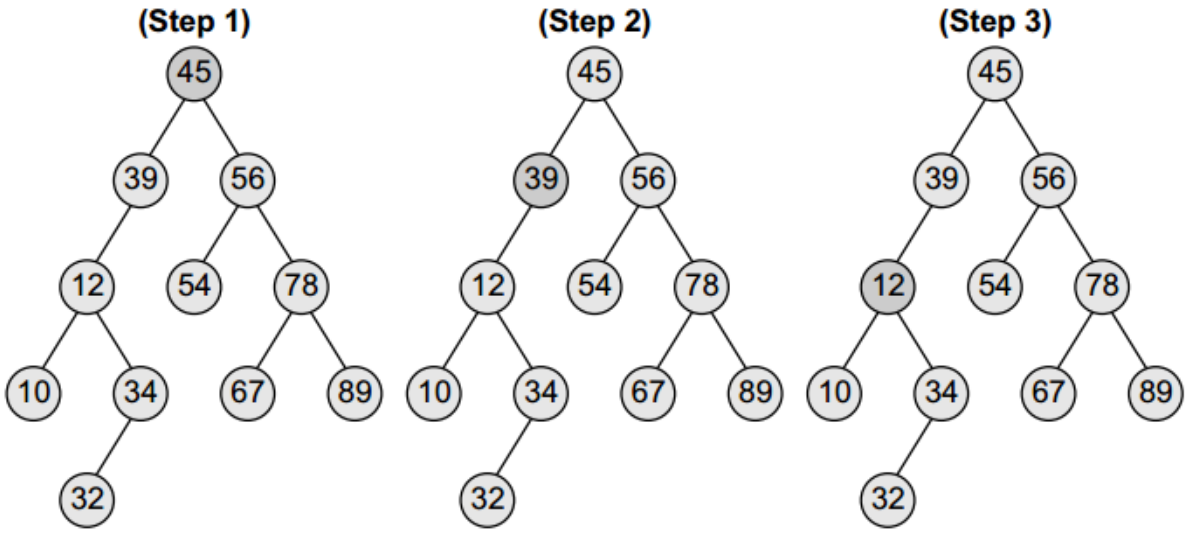


Fig: Searching a node with value 12 in the given binary search tree

Inserting a new node in binary search tree

- The insert function is used **to add a new node with a given value at the correct position** in the binary search tree.(adding the node at the correct position means that the new node should not violate the properties of binary search tree).

Algorithm to insert a given value in a binary search tree

- The initial code for the insert function is similar to the search function.
- This is because we first find the correct position where the insertion has to be done and then add the node at that position.
- The insertion function changes the structure of the tree.
- Therefore, when the insertion function is called recursively, the function should return the new tree pointer.

Insert (TREE, VAL)

```
Step 1:   IF TREE = NULL Allocate memory for TREE
            SET TREE -> DATA = VAL
            SET TREE -> LEFT = TREE -> RIGHT = NULL
          ELSE
              IF VAL < TREE DATA
                 Insert(TREE LEFT, VAL)
               ELSE
                 Insert(TREE RIGHT, VAL)
               [END OF IF]
          [END OF IF]
Step 2: END
```

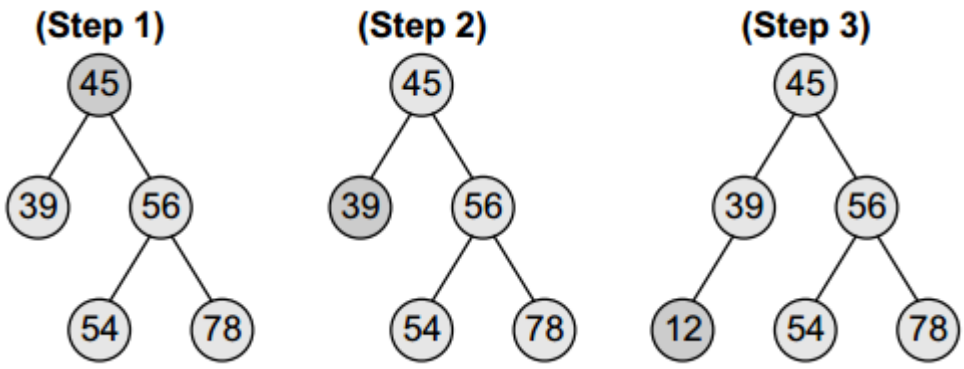


Fig: Inserting nodes with values 12

Deleting a node from a binary search tree

- The delete function B a node from the binary search tree. The properties of the binary search tree are not violated and nodes are not lost in the process.

There are 3 cases:

Case 1: deleting a node that has no children
Case 2: deleting a node with one child
Case 3: deleting a node with two children

Case 1: deleting a node that has no children(leaf node)

- This is the simplest case of deletion
- Since **no change to other nodes**

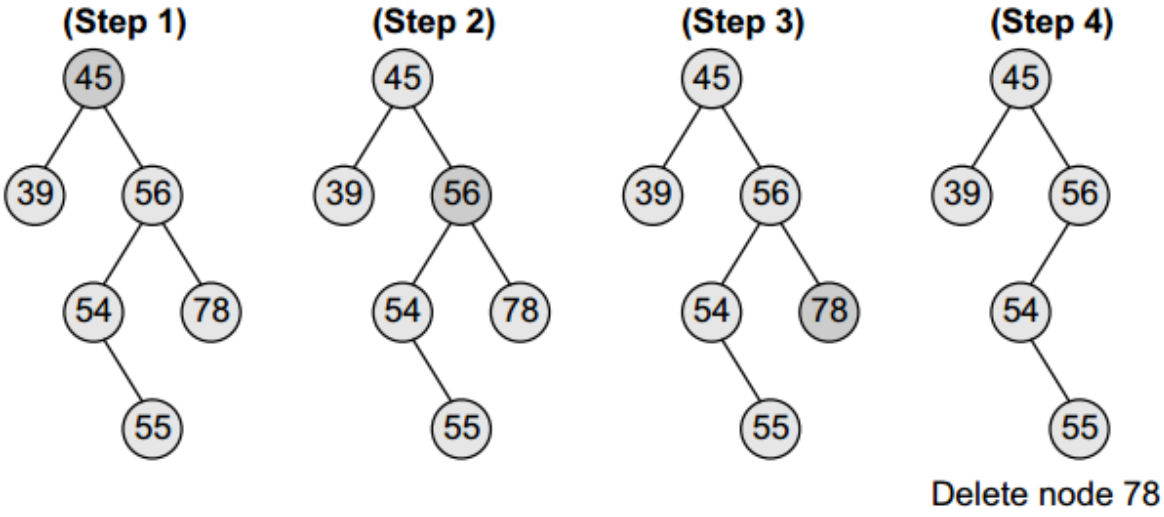


Fig: Deleting node 78 from the given binary search tree

Case 2: deleting a node with one child

- **Replace the node with its child**

- If the node is the left child of its parent, the nodes child become the left child of the nodes parent
- If the node is the right child of its parent, the nodes child becomes the right child of the nodes parent

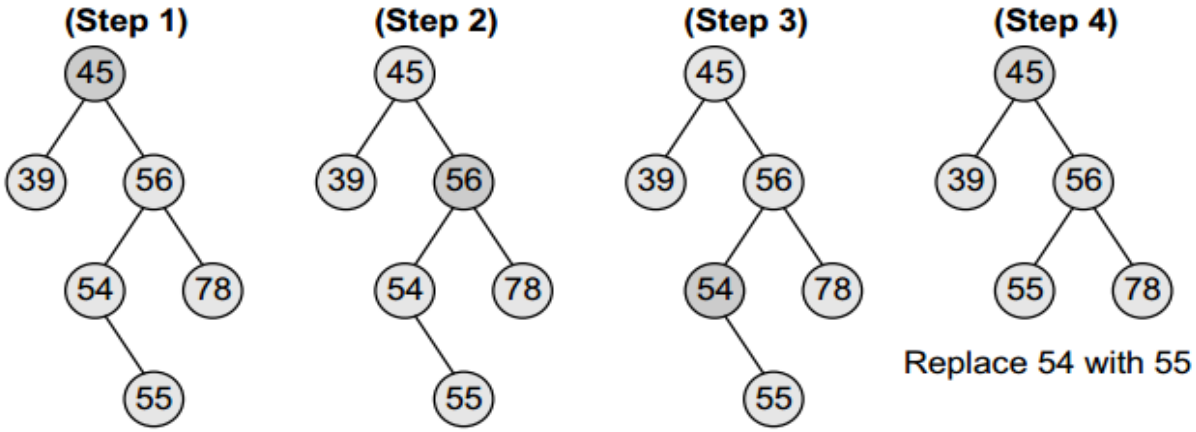


Fig: Deleting node 54 from the given binary search tree

Case 3: deleting a node with two children

- **Replace the nodes value with** its in-order predecessor**(largest value in the left sub-tree)** or in-order successor(**smallest value in the right sub-tree**)
-
- The in-order predecessor or the successor can then be deleted using any of the above cases

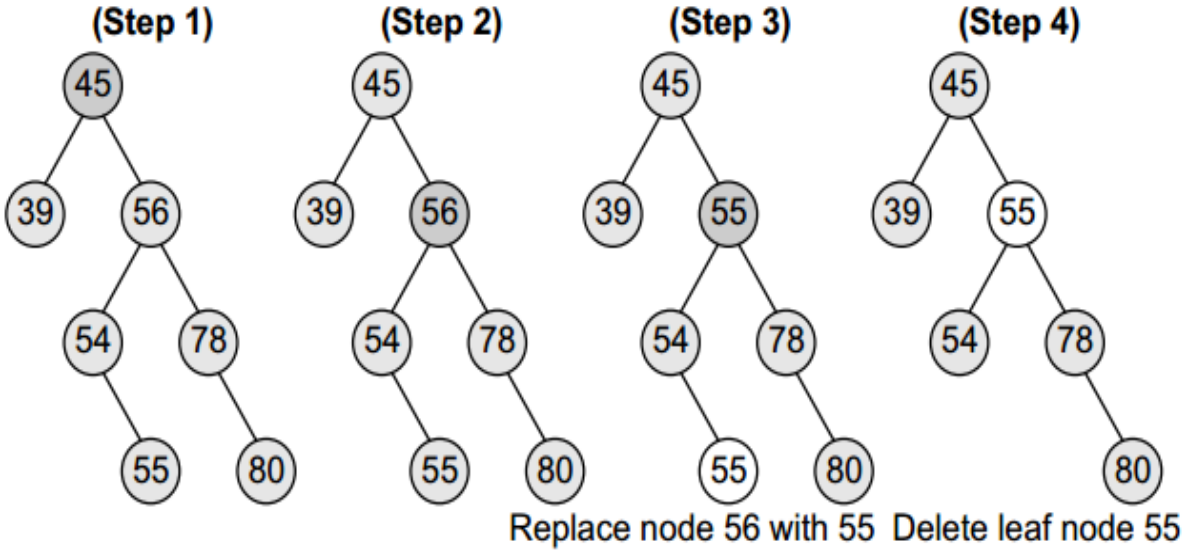


Fig: Deleting node 56 from the given binary search tree

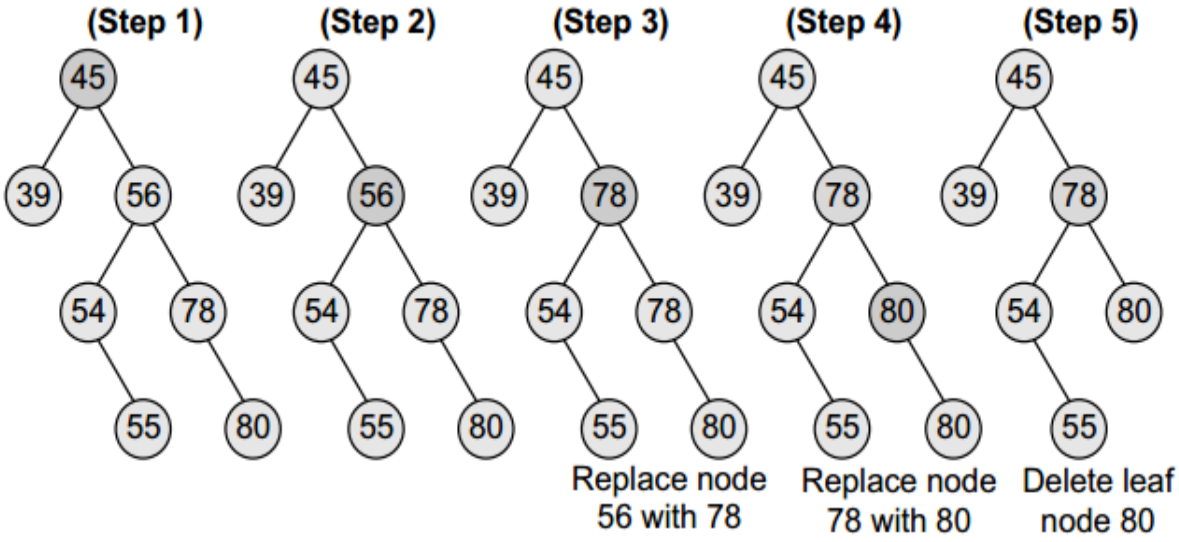


Fig: Deleting node 56 from the given binary search tree

Algorithm to delete a node from a binary search tree

Delete (TREE, VAL)

```
Step 1:  IF TREE = NULL Write "VAL not found in the tree"
         ELSE IF VAL < TREE -> DATA
                  Delete(TREE -> LEFT, VAL)
              ELSE IF VAL > TREE -> DATA
                       Delete(TREE -> RIGHT, VAL)
                   ELSE IF TREE -> LEFT AND TREE -> RIGHT
                            SET TEMP = findLargestNode(TREE -> LEFT)
                            SET TREE -> DATA = TEMP -> DATA
                            Delete(TREE -> LEFT, TEMP -> DATA)
                       ELSE
                            SET TEMP = TREE
                       IF TREE -> LEFT = NULL AND TREE -> RIGHT = NULL
                            SET TREE = NULL
                       ELSE IF TREE LEFT != NULL
                            SET -> TREE = TREE -> LEFT
                       ELSE
                            SET TREE = TREE -> RIGHT
                       [END OF IF]
                      FREE TEMP
         [END OF IF]
Step 2: END
```

Binary search tree ADT

Abstract data type binary search tree
{

Instance
Collection of nodes arranged in a hierarchical order.
A form of binary tree in which the nodes are arranged in an order.
All the nodes in the left sub-tree have a value less than the root node.
All the nodes in the right sub-tree have a value greater than the root node.
The same rule is applicable to every sub-tree in the tree.

}
{

Operations
Is empty(): return TRUE if BST is empty, return FALSE otherwise boolean data types
Insert(x) : insert element x in a node at corresponding position of BST.
Delete(x) : delete element x from a node at corresponding of BST.
Find (x) : search for a node with the value equal to x.
Preorder (): return elements in the order **NodeRoot-left-right.**
Inorder(): Return elements in the order **left- NodeRoot-Right**.
Postorder(): Return elements in the order **left- right-NodeRoot.**

}

THREADED BINARY TREES

- A threaded binary tree is the same as that of **a binary tree** but **with a difference in storing the null pointers.**
- In the linked representation, a number of nodes contain a NULL pointer, either in their left or right fields or in both.
- This space that is wasted in storing null pointer can be efficiently used to store same other useful piece of information
- The null entries can be replaced to store a pointer to the in-order previous node(predecessor) or the in-order next node(successor) of the node.

- **These special pointer are called threads and the binary trees containing threads are called threaded tree.**
- In the linked representation of a threaded binary tree **threads denoted using arrows**.

- There are many ways of threading a binary tree and each type may vary according to the way the tree is traversed. They are:
  - ❑ One – way threading
  - ❑ Two- way threading

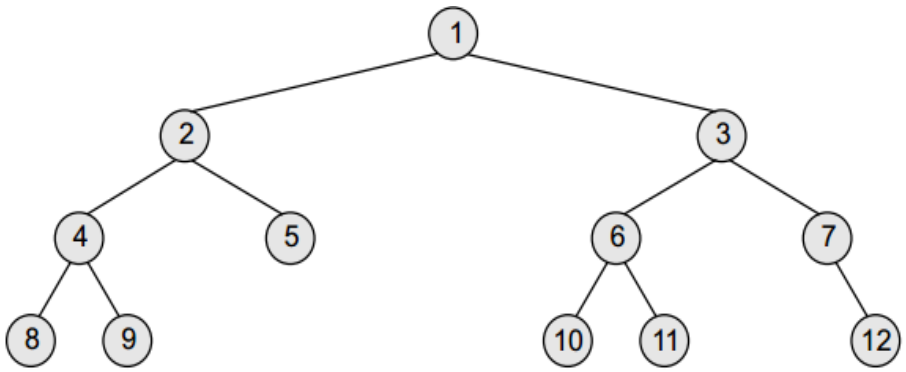


Fig: Binary tree without threading

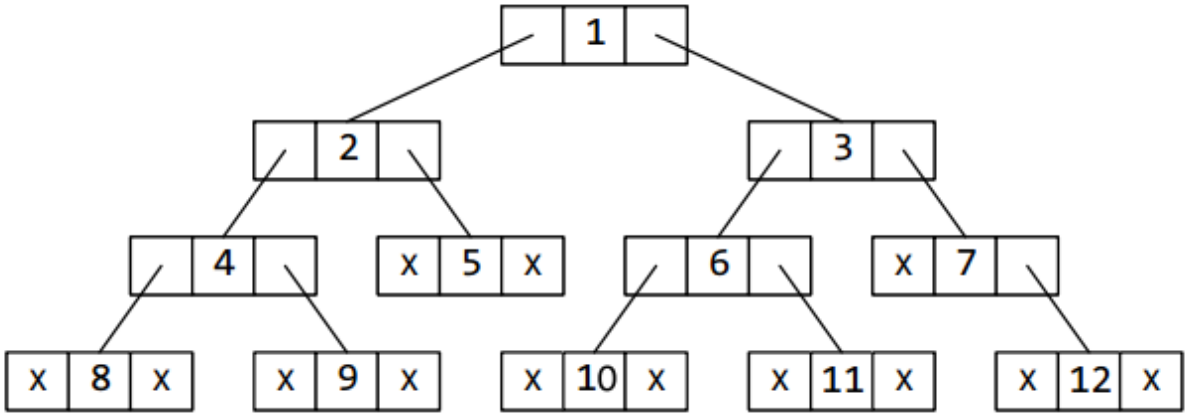


Fig: Linked representation of the binary tree (without threading)

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

One – way threading(one way threaded tree)

- In one-way **threading will appear either in the right field or the left field to the node**.
- A one-way thread tree is **also called a single threaded tree**.

- There are two types of one way threading:
  - **left- threaded binary tree.**
  - **right threaded binary tree.**

- If the thread appears in the left field, then the left field will point to the in-order predecessor of the node(in-order previous node). Such a one way threaded tree is called a **left- threaded binary tree**.

- If the thread appears in the right field, then the right field will point to the in-order successor of the node (in-order next node). Such a one-way threaded tree is called a **right threaded binary tree.**

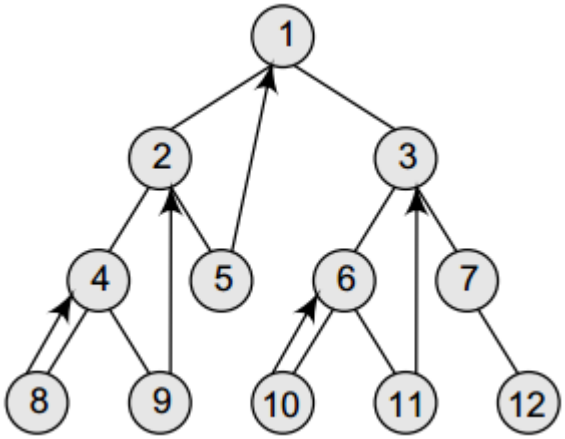


Fig: Binary tree with one-way threading

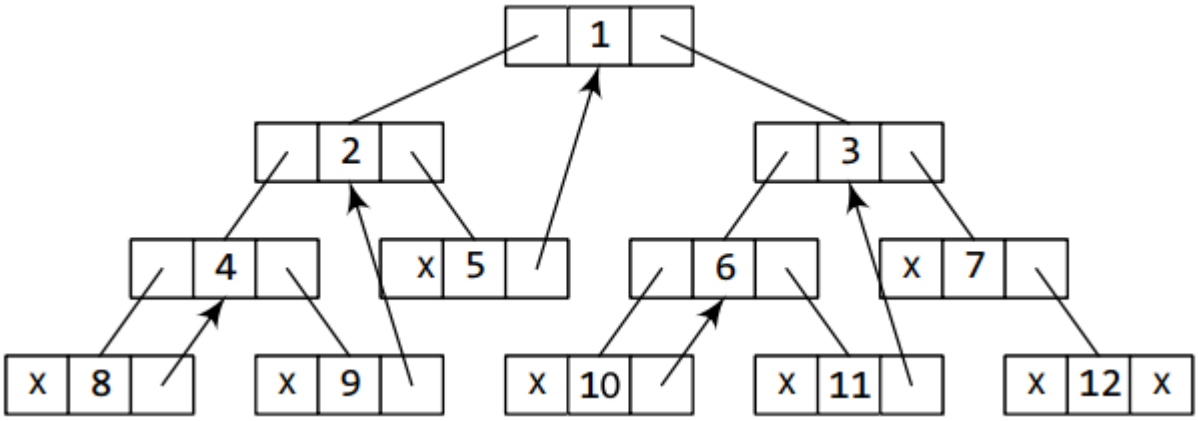


Fig: Linked representation of the binary tree with one-way threading

Two way threading(two –way threaded tree)

- In two way threading, **threads will appear in both the left and the right field of the node.**
- Two way threaded tree is **also called double threaded tree**
- The left field will point to the in-order predecessor of node(in-order previous node) and the right field will point to the in-order successor of node(in-order next)
- A two-way threaded binary tree is also a **fully threaded binary tree.**

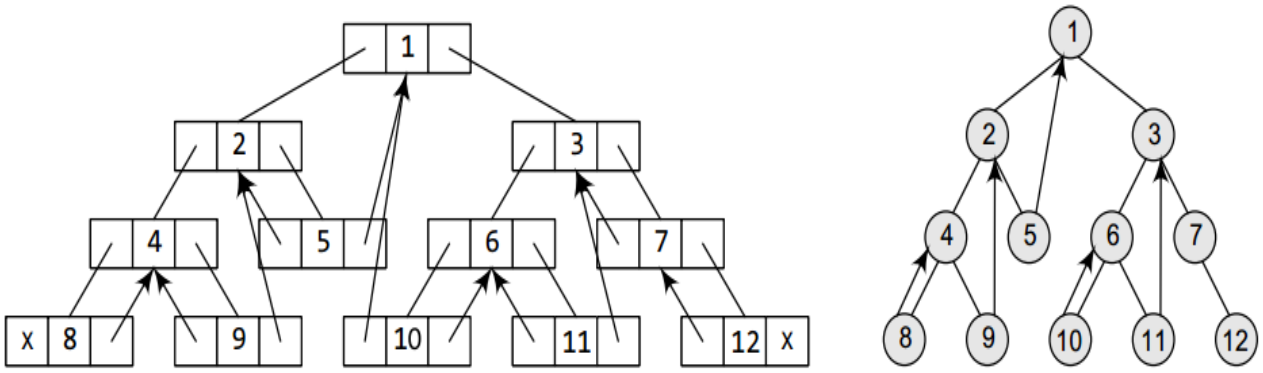


Fig: (a) Linked representation of the binary tree with threading, (b) binary tree with two-way threading

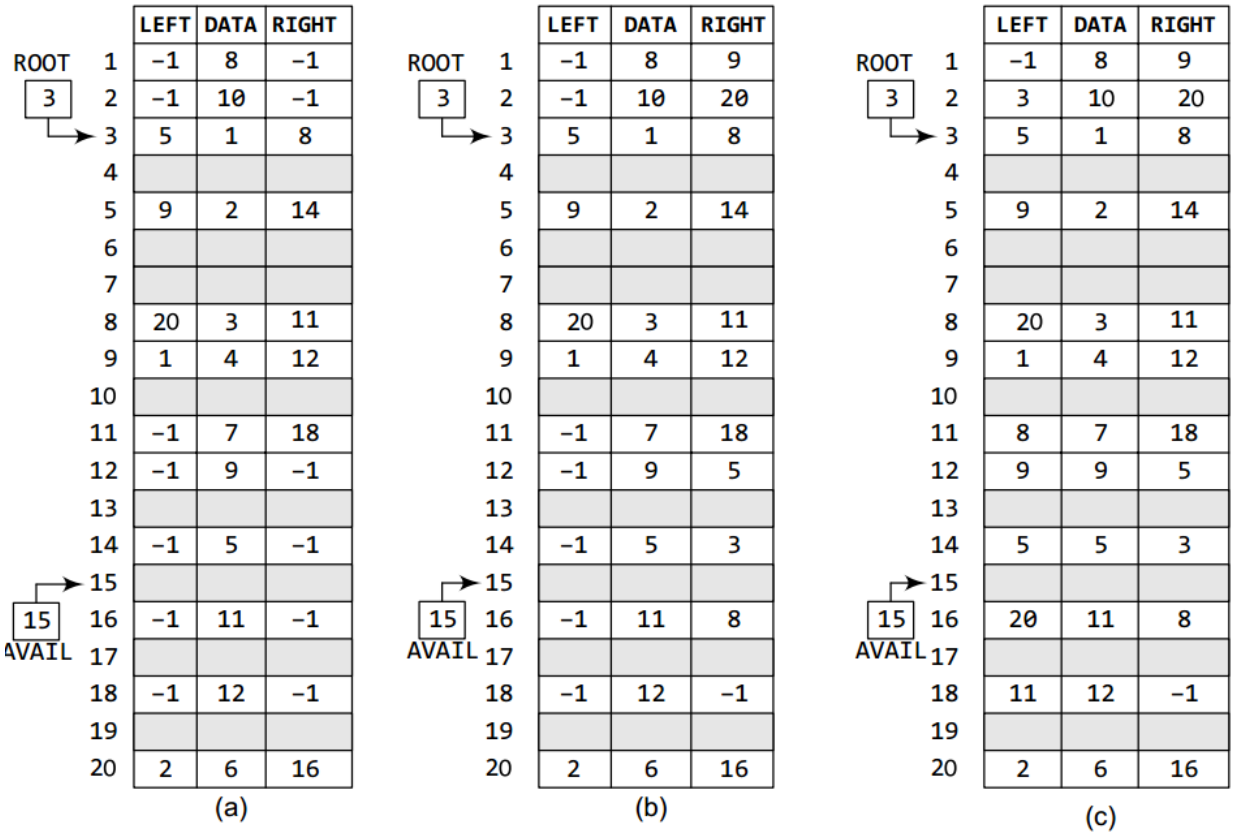


Fig: Memory representation of binary trees: (a) without threading, (b) with one-way, and (c) two-way threading.