UNIT 1 TREES

Structure

1.0 INTRODUCTION

In the previous block we discussed Arrays, Lists, Stacks, Queues and Graphs. All the data structures except Graphs are linear data structures. Graphs are classified in the non-linear category of data structures. At this stage you may recall from the previous block on Graphs that an important class of Graphs is called Trees. A Tree Is an a cyclic, connected graph. A Tree contains no loops or cycles. The concept of trees is one of the most fundamental and useful concepts in computer science. Trees have many variations, implementations and applications. Trees find their use in applications such as compiler construction, database design, windows, operating system programs, etc. A tree structure is one in which items of data are related by edges. A very common example is the ancestor tree as given in Figure 1. This tree shows the ancestors of LAKSHMI. Her parents are VIDYA and RAMKRISHNA; RAMKRISHNA'S PARENT are SUMATHI and VUAYANANDAN who are also the grand parents of LAKSHMI (on father's side); VIDYA'S parents are JAYASHRI and RAMAN and so on.

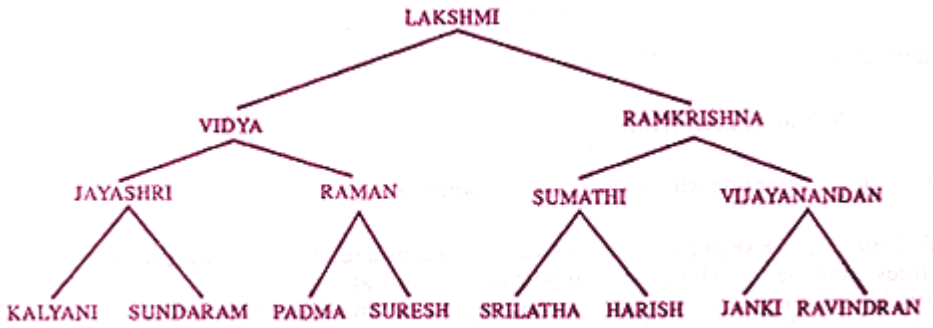


Figure 1: A Family Tree I

We can also have another form of ancestor tree as given in Figure 2.

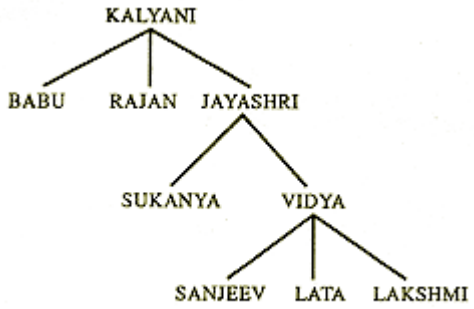


Figure 2: A Family Tree II

We could have also generated the image of tree in Figure 1 as

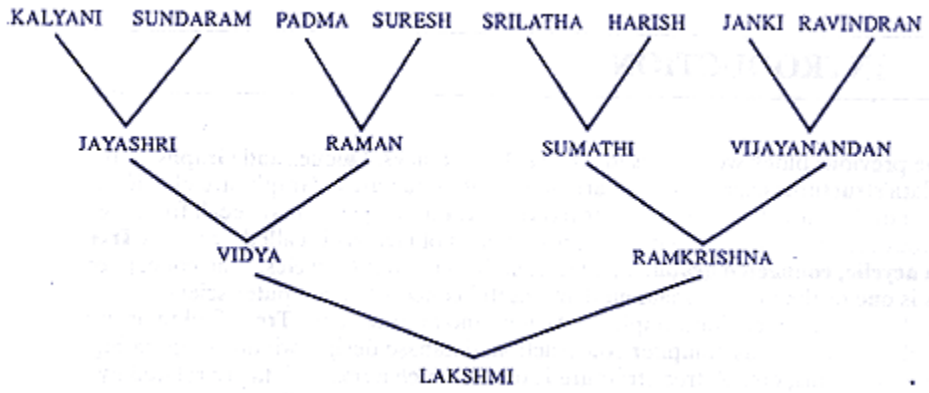


Figure 3: A Family Tree III

All the above structures are called rooted trees. A tree is said to be rooted if it has one node, called the root that is distinguished from the other nodes.

In Figure 1 the root is LAKSHMI,

In Figure 2 the root is KALYANI and

In Figure 3 the root is LAKSHMI.

In this Unit our attention will be restricted to rooted trees.

In this Unit, first we shall consider the basic definitions and terminology associated with trees, examine some important properties, and look at ways of representing trees within the computer. In later sections, we shall see many algorithms that operate on these fundamental data structures.

1.1 OBJECTIVES

At the end of this unit you shall be able to:

- define a tree, a rooted tree, a binary tree, and a binary search tree

- differentiate between a general tree and a binary tree
- describe the properties of a binary search tree
- code the insertion, deletion and searching of an element in a binary search tree
- show how an arithmetic expression may be stored in a binary tree
- build and evaluate an expression tree
- code the preorder, in order, and post order traversal of a tree

1.2 BASIC TERMINOLOGY

Trees are encountered frequently in everyday life. An example is found in the organizational chart of a large corporation. Computer Sciences in particular makes extensive use of trees. For example, in database it is useful in organizing and relating data. It is also used for scanning, parsing, generation of code and evaluation of arithmetic expressions in compiler design.

We usually draw trees with the root at the top. Each node (except the root) has exactly one node above it, which is called its parent; the nodes directly below a node are called its children. We sometimes carry the analogy to family trees further and refer to the grandparent or the sibling of a node.

Let us formally define some of the tree-related terms.

A **tree** is a non-empty collection of vertices and edges that satisfies certain . requirements.

A **vertex** is a simple object (also referred to as a node) that can have a name and can carry other associated information:

An edge is a connection between two vertices.

A tree may, therefore, be defined as a finite set of zero or more vertices such that

* there is one specially designated vertex called ROOT, and

* the remaining vertices are partitioned into a collection of sub- trees, each of which is also a tree.

In Figure 2 root is KALYANI. The three sub-trees are rooted at BABU, RAJAN and JAYASHRI. Sub-trees with JAYASHRI as root has two sub-trees rooted at SUKANYA and VIDYA and so on. The nodes of a tree have a parent-child relationship.

The root does not have a parent; but each one of the other nodes has a parent node associated to it. A node may or may not have children. A node that has no children is called a leaf node.

A line from a parent to a child node is called a branch or an edge. If a tree has n nodes, one of which is the root then there would be n-1 branches. It follows from the fact that each branch connects some node to its parent, and every node except the root has one parent.

Nodes with the same parent are called siblings. Consider the tree given in Figure 4.

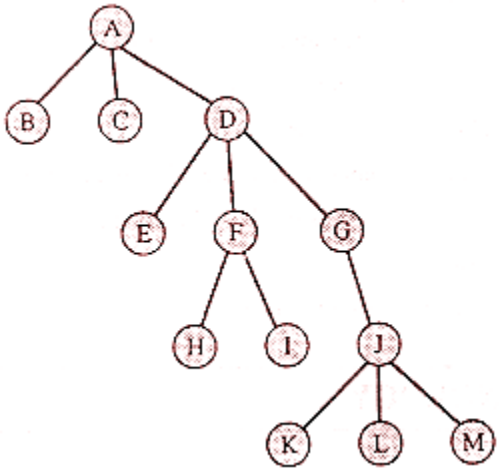


Figure 4: A Tree

K, L, and M are all siblings. B, C, D are also siblings.

A path in a tree is a list of distinct vertices in which successive vertices are connected by edges in the tree. There is exactly one path between the root and each of the other nodes in the tree. If there is more than one path between the root and some node, or if there is no path between the root and some node, then what we have is a graph, not a tree.

Nodes with no children are called leaves, or terminal nodes. Nodes with at least one child are sometimes called nonterminal nodes. We sometime refer to nonterminal nodes as internal nodes and terminal nodes as external nodes.

The length of a path is the number of branches on the path. Further if there is path from n., then n. is ancestor of i and ni. is a descendant of n.. Also there is a path of length zero from every node to itself, and there is exactly one path from the root to each node.

Let us now see how these terms apply to the tree given in Figure 4.

A path from A to K is A-D-G-J-K and the length of this path is 4.

A is ancestor of K and K is descendant of A. All the other nodes on the path are also descendants of A and ancestors of K.

The **depth** of any node n_i is the length of the path from root to n.. Thus the root is at depth 0(zero). The height of a node n. is the longest path from n_i. to a leaf. Thus all leaves are at height zero. Further, the height of a tree is same as the height of the root. For the tree in Figure 4, F is at height 1 and depth 2. D is at height 3 and depth 1. The height of the tree is 4. Depth of a node is sometimes also referred to as level of a node.

A set of trees is called a forest; for example, if we remove the root and the edges connecting it from the tree in Figure 4, we are left with a forest consisting of three trees rooted at A, D and G, as shown in Figure 5. Let us now list some of the properties of a tree:

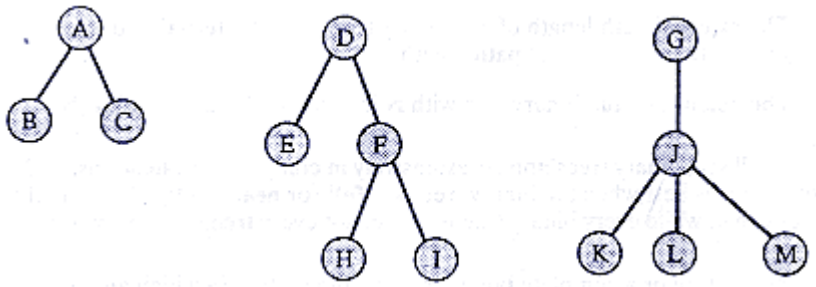


Figure 5: A Forest (sub-tree)

Properties of a tree

1. Any node can be root of the tree each node in a tree has the property that there is exactly one path connecting that node with every other node in the tree. Technically, our definition in which the root is identified, pertain to a rooted tree in which the root is not identified is called a free tree.

2. Each node, except the root ,has a unique parents and every edge connects a node to its parents .Therefor, a tree with N nodes has N-1 edges.

1.3 BINARY TREES

By definition, a **Binary tree** is a tree which is either empty or consists of a root node and two disjoint binary trees called the left subtree and right subtree. In Figure 6, a binary tree T is depicted with a left subtree, L(T) and a right subtree R(T).

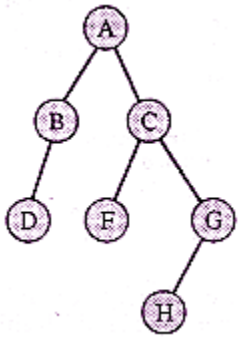


Figure 6: A Binary Tree

In a binary tree, no node can have more than two children. binary trees are special cases of general trees. The terminology we have discussed in the previous section applies to binary trees also.

Let us list the properties of binary trees:

1) Recall from the previous section the definition of internal and external nodes.- A binary tree with N internal nodes has maximum of (N + 1) **external nodes** : Root is considered as an internal node.

2) The external path length of any binary tree with N internal nodes is 2N greater than the internal path length.

3) The height of a full binary tree with N internal nodes is about $\log_2 N$

As we shall see, binary trees appear extensively in computer applications, and performance is best when the binary trees are full (or nearly full). You should note carefully that, while every binary tree is a tree, not every tree is a binary tree.

A full binary tree or a complete binary tree is a binary tree in which all internal nodes have degree and all leaves are at the same level. The figure 6a illustrates a full binary tree.

The degree of a node is the number of non empty sub trees it has. A leaf node has a degree zero.

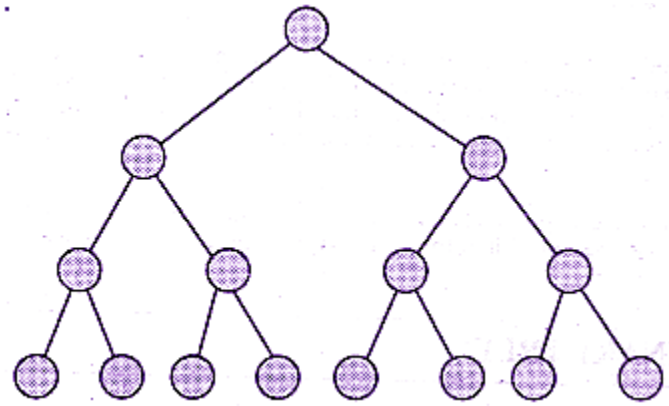


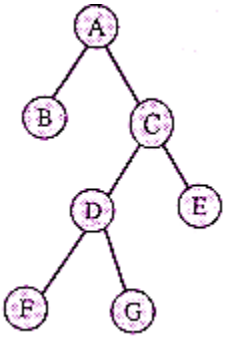
Figure 6(a): A full binary tree

Implementation

A binary tree can be implemented as an array of nodes or a linked list. The most common and easiest way to implement a tree is to represent a node as a record consisting of the data and pointer to each child of the node. Because a binary tree has at most two children, we can keep direct pointers to them. A binary tree node declaration in change may look like.

```
Typedef    Struct node
           tree_pointer
Typedef Strut node
{int data;
Tree_pointer left_child, right_child};
```

Let us now consider a special case of binary tree. it is called a 2-tree or a strictly binary tree. It is a non-empty binary tree in which either both sub trees are empty or both sub trees are 2-trees. For example, the binary trees in Figure 7(a) and 7(b) are 2-trees, but the trees in Figure 7(c) and 7(d) are not 2- trees.



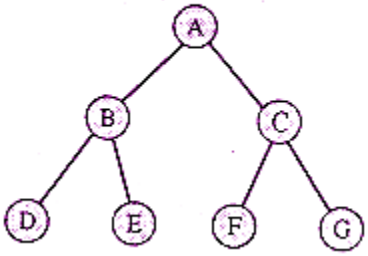


Figure : 7 (a) Figure : 7 (b)

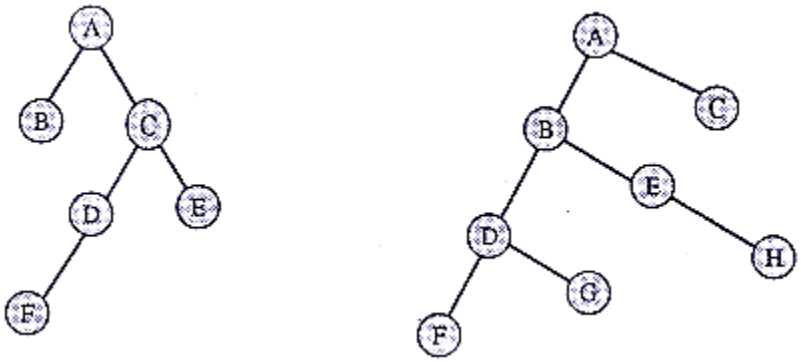


Figure 7: (a) and (b) Binary Trees(c) and (d) not Binary Trees

Binary trees are most commonly represented by linked lists. Each node can be considered as having 3 elementary fields: a data field, left pointer, pointing to left sub tree and right pointer pointing to the right sub tree.

Figure 8 contains an example of linked storage representation of a binary tree (shown in figure 6).

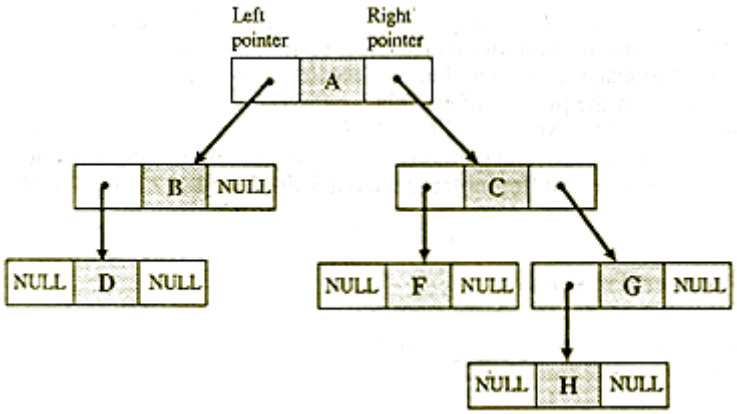


Figure 8: Linked Representation of a Binary Tree

A binary tree is said to be complete Figure 6(a) if it contains the maximum number of nodes possible for its height. In a complete binary tree:

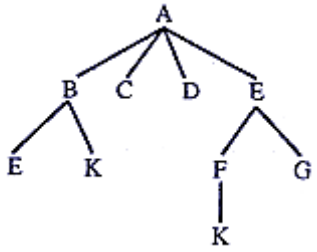
- The number of nodes at level 0 is 1.
- The number of nodes at level 1 is 2.
- The number of nodes at level 2 is 4, and so on.
- The number of nodes at level I is 2^I. Therefore for a complete binary tree with k
- levels contains $\sum_{i=0}^{k} 2^i$ nodes.

Check Your Progress 1

1. How many different trees are there with three nodes? Draw each.

---

2. Give (i) level, (ii) degree and (iii) height of each node of the following tree:



1.4 TRAVERSALS OF A BINARY TREE

Recall, from Unit 4 of Block 4, a traversal of a graph is to visit each node exactly once. In this section we shall discuss traversal of a binary tree. It is useful in many applications. For example, in searching for particular nodes. Compilers commonly build binary trees in the process of scanning, parsing, generating code and evaluation of arithmetic expression. Let T be a binary tree. There are a number of different ways to proceed. The methods differ primarily in the order in which they visit the nodes. The four different traversals of T are In order, Post order, Preorder and Level-by-level traversal.

1.4.1 In order Traversal

It follows the general strategy of Left-Root-Right. In this traversal, if T is not empty, we first traverse (in order) the left sub tree;

then visit the root node of T, and

then traverse (in order) the right sub tree.

Consider the binary tree given in Figure 9.

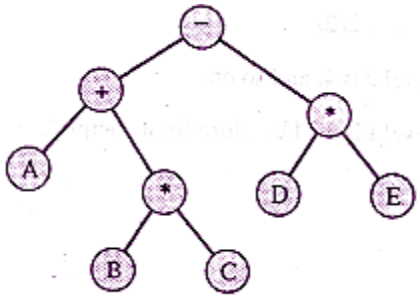


Figure 9. Expression Tree

This is an example of an expression tree for (A + B*C)-(D*E)

A binary tree can be used to represent arithmetic expressions if the node value can be either operators or operand values and are such that:

* each operator node has exactly two branches

* each operand node has no branches, such trees are called expression trees.

Tree, T, at the start is rooted at '_';

Since left(T) is not empty; current T becomes rooted at +;

Since left(T) is not empty; current T becomes rooted at 'A'.

Since left(T) is empty; we visit root i.e. A.

We access T' root i.e. '+'.

We now perform in order traversal of right(T).

Current T becomes rooted at '*'.

Since left(T) is not empty; Current T becomes rooted at 'B' since left(T) is empty; we visit its root i.e. B; cheek for right(T) which is empty, therefore, we move back to parent tree. We visit its root i.e. '*'.

Now in order traversal of right(T)is performed; which would give us 'C'. We visit T's root i.e. 'D' and perform in order traversal of right(T); which would give us'* and E'. Therefore, the complete listing is

A + B * C - D * E

You may note that expression is in infix notation. The in order traversal produces a(parenthesized) left expression, then prints out the operator at root and then a(parenthesized) right expression. This method of traversal is probably the most widely used. The following is a C function for in order traversal of a binary tree

```
procedure  VOID INORDER (TREE_ POINTER PTR)

    /* inorder tree traversal */
    { if (ptr) { inorder ptr ----left_child);
    Printif ("%d" , ptr---data);
    Inorder (ptr---right_child);
    }
}
```

Figure 10 gives a trace of the in order traversal of tree given in figure 9.

Root of the tree	Output
-	
+	
A	
Empty sub tree	A
Empty sub tree	+
*	
B	
Empty sub tree	B

Empty sub tree	*
C	
Empty sub tree	C
Empty sub tree	-
*	
D	
Empty sub tree	D
Empty sub tree	*
E	
Empty sub tree	E
Empty sub tree	over

Figure 10: Trace of in order traversal of tree given in figure 9

Please notice that this procedure, like the definition for traversal is recursive.

1.4.2 Post order Traversal

In this traversal we first traverse left(T) (in post order); then traverse Right(T) (in post order); and finally visit root. It is a Left-Right-Root strategy, i.e.

Traverse the left sub tree In Post order.

Traverse the right sub tree in Post order.

Visit the root.

For example, a post order traversal of the tree given in Figure 9 would be

A B C * + D E * -

You may notice that it is the postfix notation of the expression

(A + (B*C)) -(D*E)

We leave the details of the post order traversal method as an exercise. You may also implement it using Pascal or C language.

1.4.3 Preorder Traversal

In this traversal, we visit root first; then recursively perform preorder traversal of Left(T); followed by pre order. traversal of Right(T) i.e. a Root-Left-Right traversal, i.e.

Visit the root

Traverse the left sub tree preorder.

Traverse the right sub tree preorder.

A preorder traversal of the tree given in Figure 9 would yield

- +A*BC*DE

It is the prefix notation of the expression

(A+ (B*C)) - (D*E)

Preorder traversal is employed in Depth First Search. (See Unit 4, Block 4). For example, suppose we make a depth first search of the binary tree given in Figure 11.

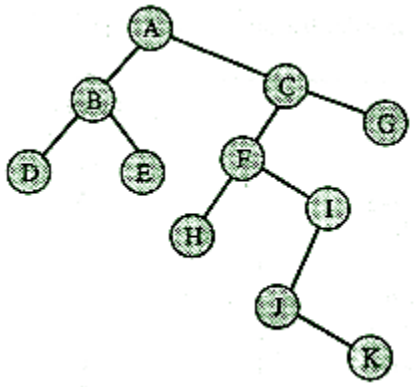


Figure 12: Binary tree example for depth first search

We shall visit a node; go left as deeply as possible before searching to its right. The order in which the nodes would be visited is

A B D E C F H I J K G

which is same as preorder traversal.

1.4.4 Level by Level traversal

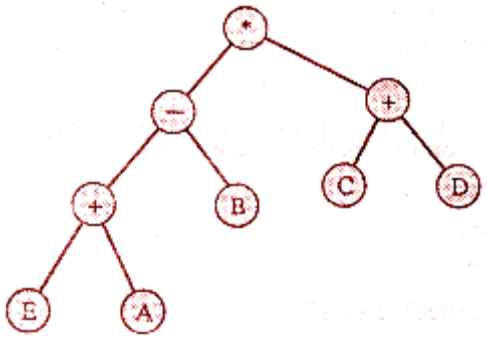
In this method we traverse level-wise i.e. we first visit node root at level 'O' i.e. root. There would be just one. Then we visit nodes at level one from left to right. There would be at most two. Then we visit all the nodes at level '2' from left to right and so on. For example the level by level traversal of the binary tree given in Figure 11 will yield

A B C D E F G H I J K

This is same as breadth first search (see Unit 4, Block 4). This traversal is different from other three traversals in the sense that it need not be recursive, therefore, we may use queue kind of a data structure to implement it, while we need stack kind of data structure for the earlier three traversals.

Check Your Progress 2

Question 1: Traverse the tree given below in preorder, in order, post order and level by level giving a list of nodes visited.



1.5 BINARY SEARCH TREES (BST)

A Binary Search Tree, BST, is an ordered binary tree T such that either it is an empty tree or

- each data value in its left sub tree less than the root value,
- each data value. in its right sub tree greater than the root value, and
- left and right sub trees are again binary search trees.

Figure 12(a) depicts a binary search tree, while the one in

Figure 12(b) is not a binary search tree.

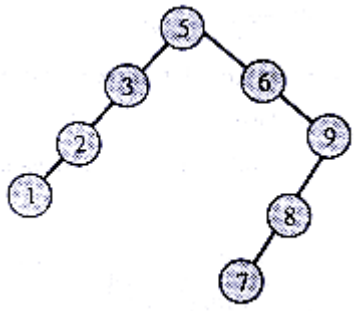


Figure 12(a): Binary Search Tree

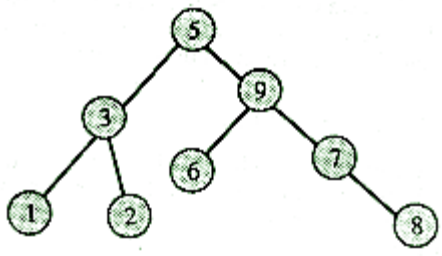


Figure 12(b): Binary tree but not binary search tree

Clearly, duplicate items are not allowed in a binary search tree. You may also notice that an in order traversal of a BST yields a sorted list in ascending order.

1.5.1 Operations on a BST

We now give a list of the operations that are usually performed on a BST.

1. Initialization of a **BST:** This operation makes an empty tree.

2. Cheek whether **BST** is Empty: This operation cheeks whether the tree is empty.

3. Create a node for the **BST:** This operation allocates memory space for the new node; returns with error if no spade is available.

4. Retrieve a node's data.

5. Update a node's data.

6. Insert a node in **BST.**

7. Delete a node (or sub tree) of a **BST**.

8. Search for a node in **BST**.

9. Traverse (in inorder, preorder, or post order) a **BST**.

We shall describe some of the operations in detail.

1.5.2 Insertion in a BST

Inserting a node to the tree: To insert a node in a BST, we must check whether the tree already contains any nodes. If tree is empty, the node is placed in the root node. If the tree is not empty, then the proper location is found and the added node becomes either a left or a right child of an existing node. The logic works this way:

```
add-node, (node, value)
{
        if (two values are same)
        {
                duplicate.,
                return (FAILURE)
            }
    else if (value
    {
        if (left child exists)
        {
                add-node (left child, value);
            }
            else
            {
                allocate new node and make left
                child point to it,.
                return (SUCCESS);
            }
```

```
        }


else if (value value stored in current node)
{
        if (right child exists)
        {
                add-node (right child, value);
            }
            else
            {
                allocate new node and make right child
                point to it
                return (SUCCESS);
            }
    }
    }
```

The function continues recursively until either it finds a duplicate (no duplicate strings are allowed) or it hits a dead end. If it determines that the value to be added belongs to the left-child sub tree and there is no left-child node, it creates one. If a left-child node exists, then it begins its search with the sub tree beginning at this node. If the function determines that the value to be added belongs to the right of the current node, a similar process occurs.

Let us consider a BST given in Figure 13(a).

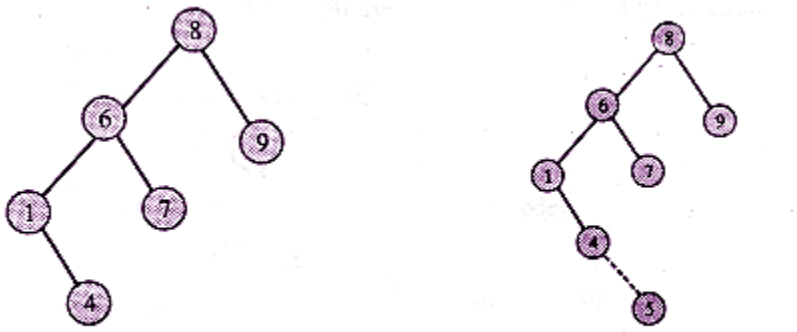


Figure 13: Insertion In a Binary Search Tree

If we want to insert 5 in the above BST, we first search the tree. If the key to be inserted is found in tree, we do nothing (since duplicates are not allowed), otherwise a nil is returned. In case a nil is returned, we insert the data at the last point traversed. In the example above a search operation will return nil on not finding a right, sub tree of tree rooted at 4. Therefore, 5 must be inserted as a right child of 4.

1.5.3 Deletion of a node

Once again the node to be deleted is searched in BST. If found, we need to consider the following possibilities:

(i) If node is a leaf, it can be deleted by making its parent pointing to nil. The deleted node is now unreferenced and may be disposed off.

(ii) If the node has one child; its parent's pointer needs to be adjusted. For example for node 1 to be deleted from BST given in Figure 13(a); the left pointer of node 6 is made to point to child of node 1 i.e. node 4 and the new structure would be

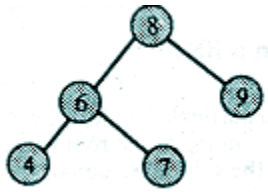


Figure 14: Deletion of a Terminal Node

(iii) If the node to be deleted has two children; then the value is replaced by the smallest value in the right sub tree or the largest key value in the left sub tree; subsequently the empty node is recursively deleted. Consider the BST in Figure 15.

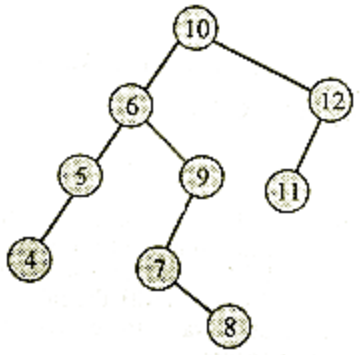


Figure 15: Binary search tree

If the node 6 is to be deleted then first its value is replaced by smallest value in its right subtree i.e. by 7. So we will have Figure 16.

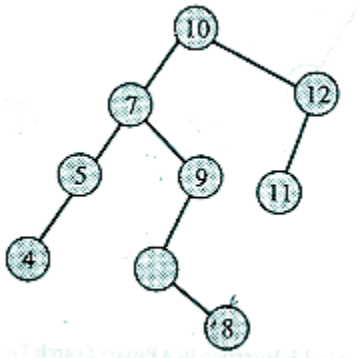


Figure 16. Deletion of a nod having left and right child

Now we need to, delete this empty node as explained in (iii).

(iv) Therefore, the final structure would be Figure 17.

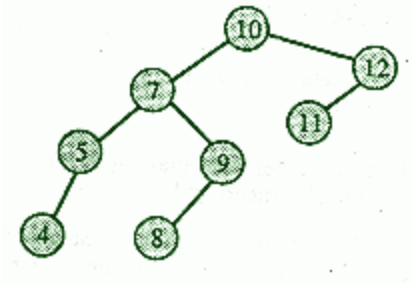


Figure 17: Tree alter a deletion of a node having left and right child

1.5.4 Search for a key in a BST

To search the binary tree for a particular node, we use procedures similar to those we used when adding to it. Beginning at the root node, the current node and the entered key are compared. If the values are equal success is output. If the entered value is less than the value in the node, then it must be in the left-child sub tree. If there is no left-child sub tree, the value is not in the tree i.e. a failure is reported. If there is a left-child subtree, then it is examined the same way. Similarly, it the entered value is greater than the value in the current node, the right child is. searched. Figure 18 shows the path through the tree followed in the search for the key H.

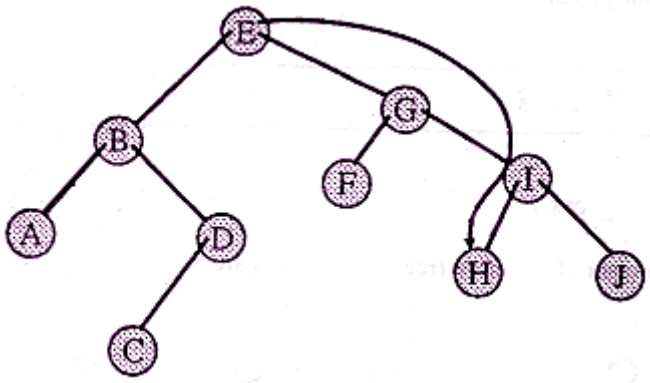


Figure 18: Search for a Key In BST

```
find-key (key value, node)
{
if (two values are same)
{
      print value stored in node;
      return (SUCCESS);
}
else if (key value value stored in current node)
{
                  if (left child exists)
                  {
                        find-key (key-value, left hand);
                  }
                  else
                  {
                        there is no left subtree.,
```

```
                    return (string not found)
                }
        }
        else if (key-value value stored in current node)
        {
            if (right child exists)
        {
            find-key (key-value, rot child);
        }
        else
        {
            there is no right subtree;
            return (string not found)
        }
    },
}
```

1.6 SUMMARY

This unit introduced the tree data structure which is an **acyclic, connected, simple** graph. Terminology pertaining to trees was introduced. A special case of general case of general tree, a binary tree was focussed on. In a binary tree, each node has a maximum of two subtrees, left and right subtree. Sometimes it is necessary to traverse a tree, that is, to visit all the tree's nodes. Four methods of tree traversals were presented in order, post order, preorder and level by level traversal. These methods differ in the order in which the root, left subtree and right subtree are traversed. Each ordering is appropriate for a different **type of applications.**

An important class of binary trees is a complete or full binary tree. A full binary tree is one in which internal nodes completely fill every level, except possibly the. last. A complete binary tree where the internal nodes on the bottom level all appear to the left of the external nodes on that level. Figure 6a shows an example of a complete binary tree.

1.7 MODEL ANSWERS

Check Your Progress 1

1. The Total number of different trees with 3 nodes are 5

2.

Node	Level	Out degree	In degree
A	0	4	0
B	1	2	1
C	1	0	1
D	1	0	1

E	2	0	1
k	2	0	1
l	1	2	1
f	2	1	1
g	2	0	1
h	3	0	1

Check Your Progress 2

Answer:

Preorder:	*-+EAB+CD
Inorder:	E+A-B*C+D
Postorder:	EA+B-CD + *-
Level by level:	*- + +BCDEA