MODULE IV NOTES

Module IV : Trees

Contents

Credits:GAT

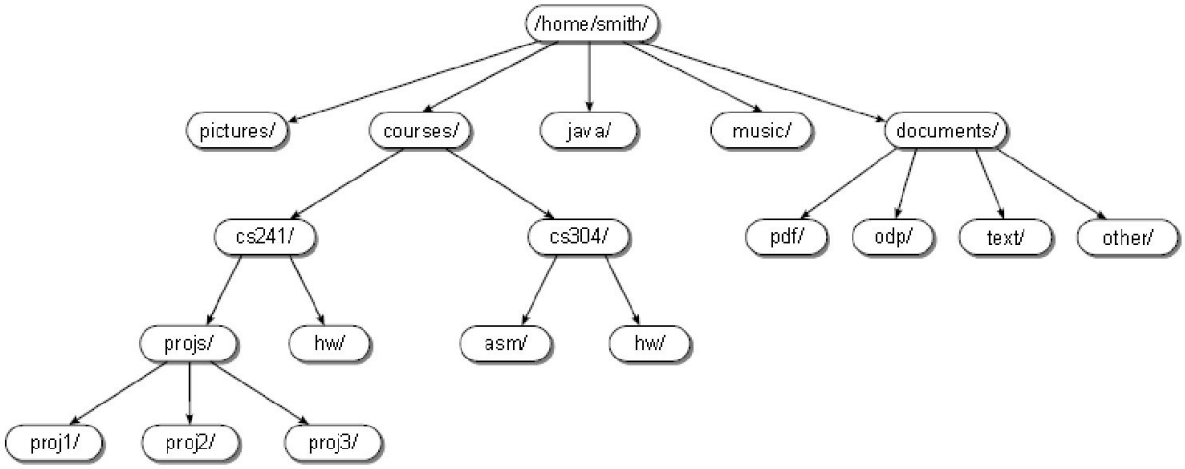
Module IV :Trees

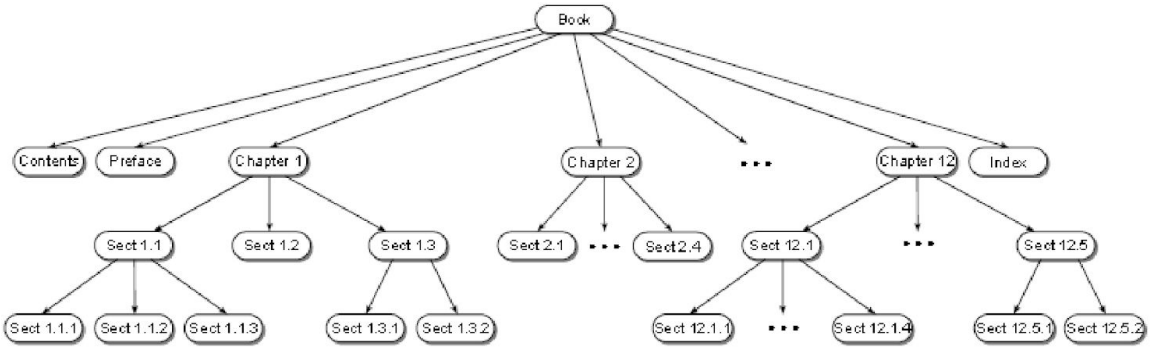
1. Introduction

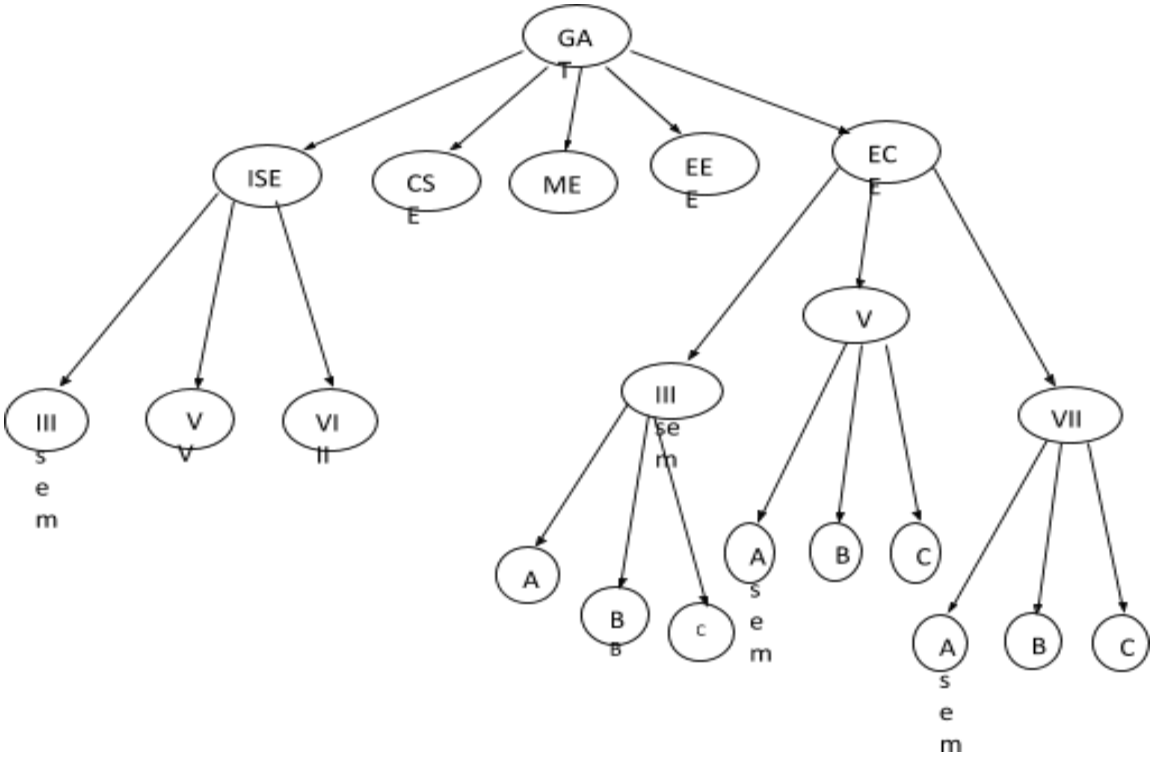
A tree structure consists of nodes and edges that organize data in a hierarchical fashion. The relationships between data elements in a tree are similar to those of a family tree: “child," “parent”, "ancestor," etc. The data elements are stored in **nodes** and pairs of nodes are connected by **edges**. The edges represent the relationship between the nodes that are linked with arrows or directed edges to form a ***hierarchical structure*** resembling an upside-down tree complete with **branches, leaves**, and even a **root.**

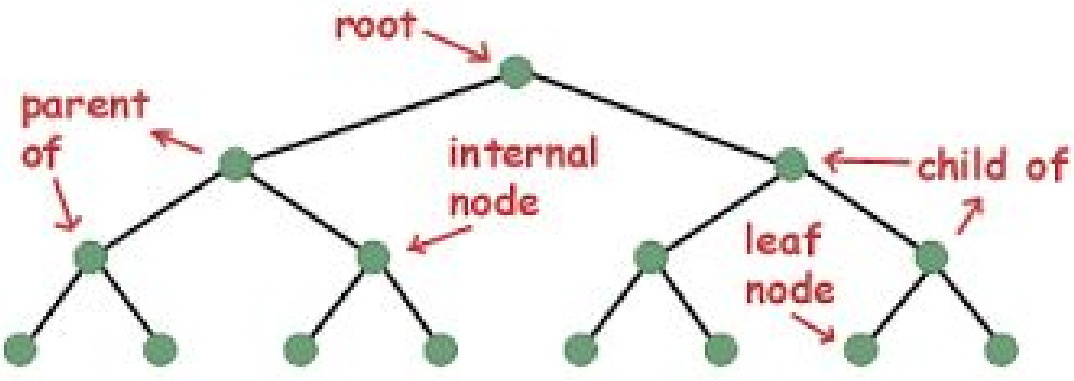
Formally, we can define a tree as a set of nodes that either is empty or has a node called the root that is connected by edges to zero or more subtrees to form a hierarchical structure. Each subtree is itself by definition a tree. A classic example of a tree structure is the representation of directories and subdirectories in a UNIX file system. The top tree in Figure 13.1 illustrates the hierarchical nature of a student's home directory in the UNIX file system. Trees can be used to represent structured data, which results in the subdivision of data into smaller and smaller parts.

A simple example of this use is the division of a book into its various parts of chapters, sections, and subsections, as illustrated by the bottom tree in Figure 1. Trees are also used for making decisions. One that you are most likely familiar with is the phone, or menu, tree. When you call customer service for most businesses today, you are greeted with an automated menu that you have to traverse. The various menus are nodes in a tree and the menu options from which you can choose are branches to other nodes. Some of the examples are shown below.




Book
Contents
Preface
Chapter 1
Chapter 2
Chapter 12
Index
Sect 1.1
Sect 1.2
Sect 1.3
Sect 2.1
Sect2.4
Sect 12.1
Sect 12.5
Sect 1.1.1
Sect 1.1.2
Sect 1.1.3
Sect 1.3.1
Sect 1.3.2
Sect 12.1.1
Sect 12.1.4
Sect 12.5.1
Sect 12.5.2


GA
ISE
CS
ME
EE
EC
III
V
VI
V
III
VII
A
B
C
A
B
C
A
B
C

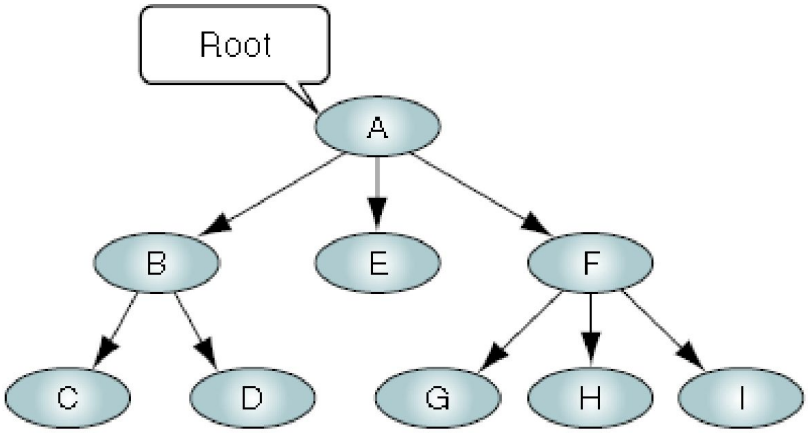

root
parent
of
internal
node
child of
leaf
node

2. Concept of Tree

It implies that we organize the data so that items of information are related by the branches.

Definition: A tree is a finite set of one or more nodes such that:

1. There is a specially designated node called the root.
2. The remaining nodes are partitioned into n ≥ 0 disjoint sets $T1,\ldots, T_n$, where each of these sets is a tree. We call $T1,\ldots, T_n$ the subtrees of the root.



Binary Tree Representation in memory Using Doubly Linked List

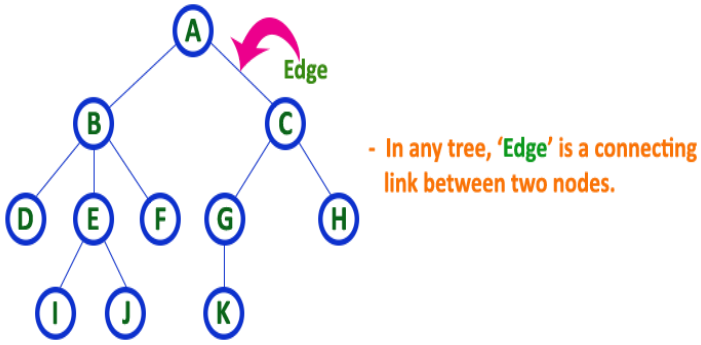
3. Tree Terminology

Root of the tree:

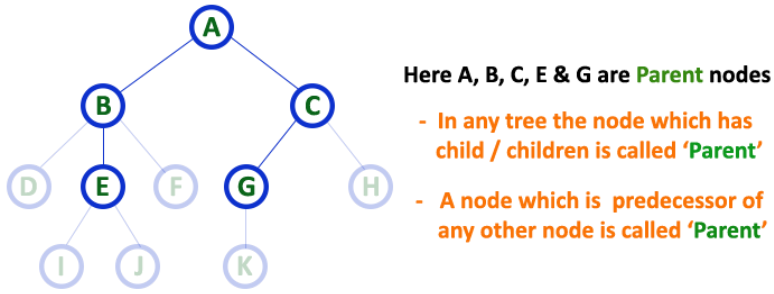
In a tree data structure, the first node is called as Root Node. Every tree must have root node. We can say that root node is the origin of tree data structure. In any tree, there must be only one root node. We never have multiple root nodes in a tree.

Node: It is stands for the item of information and the branches to other nodes.

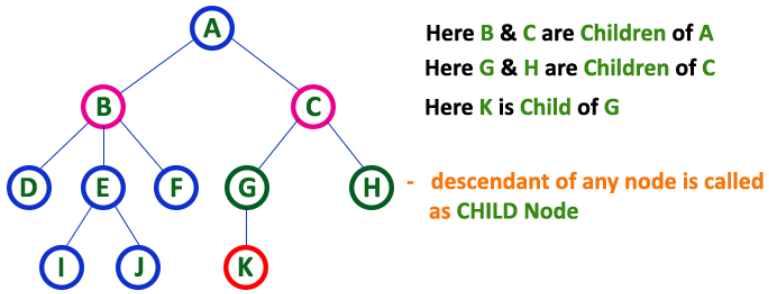
Edge : In a tree data structure, the connecting link between any two nodes is called as EDGE. In a tree with 'N' number of nodes there will be a maximum of 'N-1' number of edges.



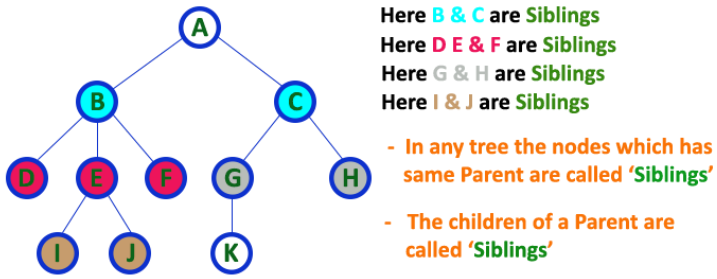
Parent: node: a node that has subtrees is the parent of the roots of the subtrees. In a tree data structure, the node which is predecessor of any node is called as PARENT NODE. In simple words, the node which has branch from it to any other node is called as parent node. Parent node can also be defined as "The node which has child / children".



Child node: a node that is the roots of the subtrees are the children of the node. In a tree data structure, the node which is descendant of any node is called as **CHILD Node**. In simple words, the node which has a link from its parent node is called as child node. In a tree, any parent node can have any number of child nodes. In a tree, all the nodes except root are child nodes.

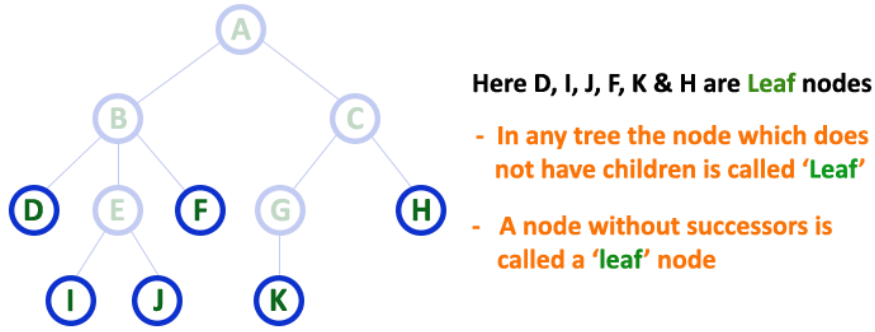


Siblings: In a tree data structure, nodes which belong to same Parent are called as **SIBLINGS**. In simple words, the nodes with same parent are called as Sibling nodes.



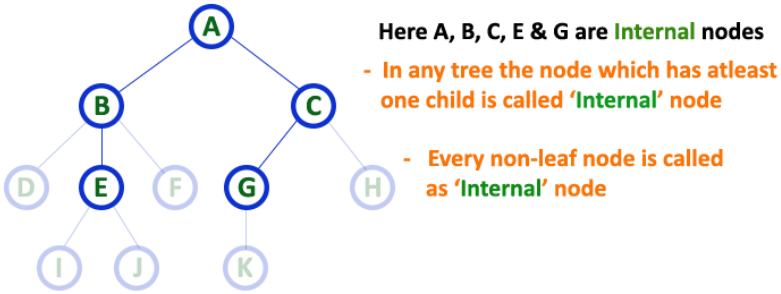
Leaf: In a tree data structure, the node which does not have a child is called as **LEAF Node**. In simple words, a leaf is a node with no child.

In a tree data structure, the leaf nodes are also called as **External Nodes**. External node is also a node with no child. In a tree, leaf node is also called as '**Terminal**' node.



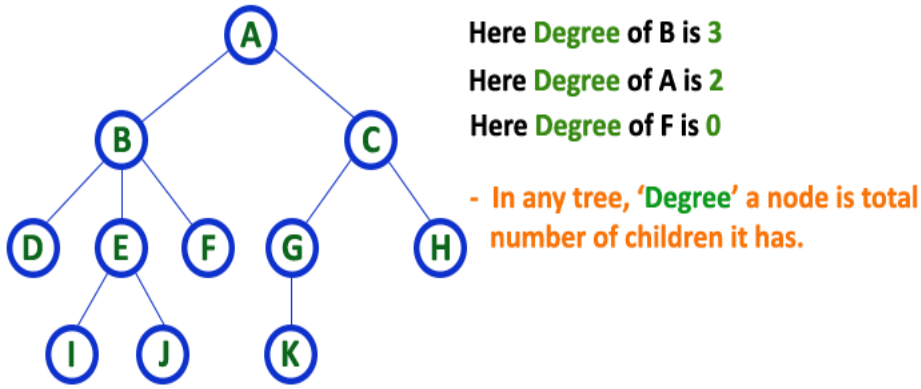
Internal Nodes: In a tree data structure, the node which has atleast one child is called as **INTERNAL Node**. In simple words, an internal node is a node with atleast one child.

In a tree data structure, nodes other than leaf nodes are called as **Internal Nodes**. **The root node is also said to be Internal Node** if the tree has more than one node. Internal nodes are also called as '**Non-Terminal**' nodes.



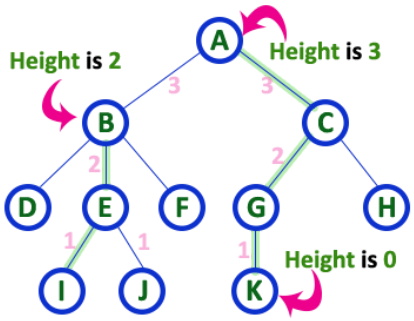
Degree: The degree of a node: It is the number of subtrees of the node.

In a tree data structure, the total number of children of a node is called as **DEGREE** of that Node. In simple words, the Degree of a node is total number of children it has. The highest degree of a node among all the nodes in a tree is called as '**Degree of Tree**'



The degree of a tree: It is the maximum degree of the nodes in the tree

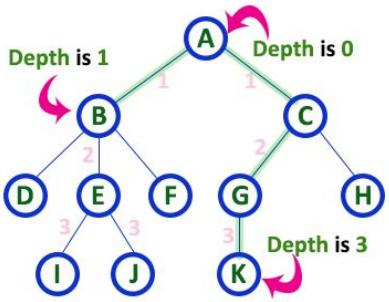
Height: In a tree data structure, the total number of egdes from leaf node to a particular node in the longest path is called as **HEIGHT** of that Node. In a tree, height of the root node is said to be **height of the tree**. In a tree, **height of all leaf nodes is '0'.**



Here Height of tree is 3

- In any tree, 'Height of Node' is total number of Edges from leaf to that node in longest path.
- In any tree, 'Height of Tree' is the height of the root node.

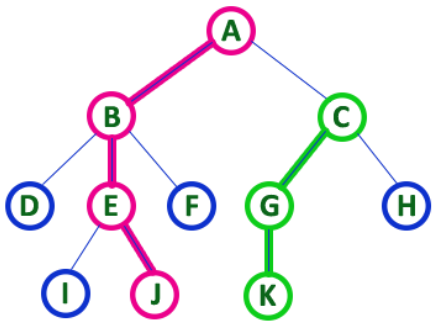
Depth: In a tree data structure, the total number of egdes from root node to a particular node is called as **DEPTH** of that Node. In a tree, the total number of edges from root node to a leaf node in the longest path is said to be **Depth of the tree**. In simple words, the highest depth of any leaf node in a tree is said to be depth of that tree. In a tree, **depth of the root node is '0'.**



Here Depth of tree is 3

- In any tree, 'Depth of Node' is total number of Edges from root to that node.
- In any tree, 'Depth of Tree' is total number of edges from root to leaf in the longest path.

Path: In a tree data structure, the sequence of Nodes and Edges from one node to another node is called as **PATH** between that two Nodes. **Length of a Path** is total number of nodes in that path. In below example **the path A - B - E - J has length 4**.



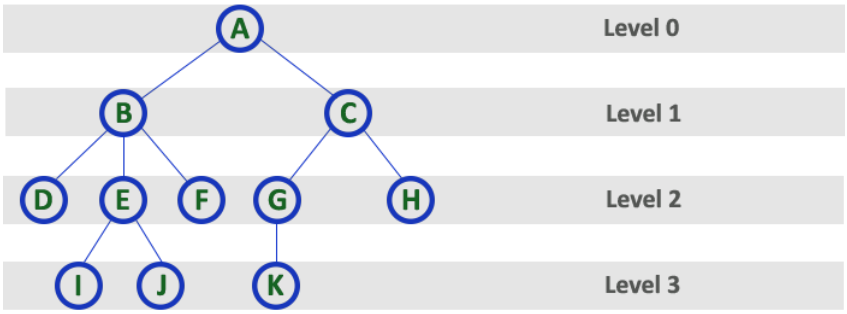
- In any tree, 'Path' is a sequence of nodes and edges between two nodes.

Here, 'Path' between A & J is
A - B - E - J

Here, 'Path' between C & K is
C - G - K

Level of the tree: We define the level of a node by initially letting the root be at level one.

In a tree data structure, the root node is said to be at Level 0 and the children of root node are at Level 1 and the children of the nodes which are at Level 1 will be at Level 2 and so on... In simple words, in a tree each step from top to bottom is called as a Level and the Level count starts with '0' and incremented by one at each level (Step).



The depth of a tree: It also called height of a tree. It is the maximum level of any node in the tree

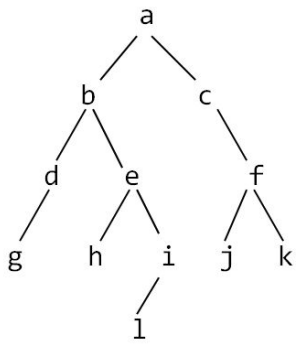
Node A is the parent of node B if node B is a child of A

Node A is an ancestor of node B if A is a parent of B, or if some child of A is an ancestor of B

In less formal terms, A is an ancestor of B if B is a child of A, or a child of a child of A, or a child of a child of a child of A, etc.

Node B is a descendant of A if A is an ancestor of B

Nodes A and B are siblings if they have the same parent

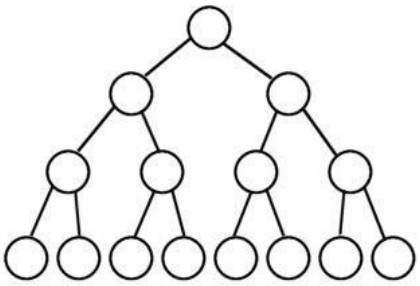
	The size of a binary tree is the number of nodes in it This tree has size 12 The depth of a node is its distance from the root a is at depth zero e is at depth 2 The depth of a binary tree is the depth of its deepest node This tree has depth 4

4. Types of Trees

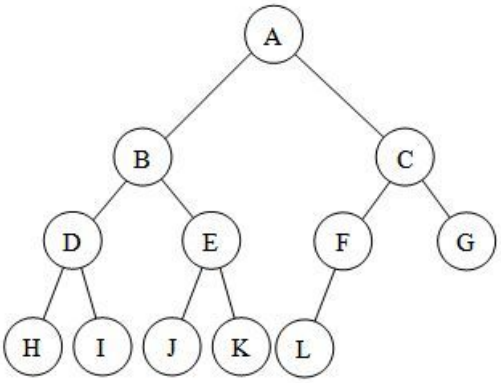
General Tree, Full Tree, Complete Tree, Binary Tree, Binary Search Tree, Heap

1. A **complete binary tree** (sometimes proper binary tree or 2-tree) is a tree in which every node other than the leaves has two children.

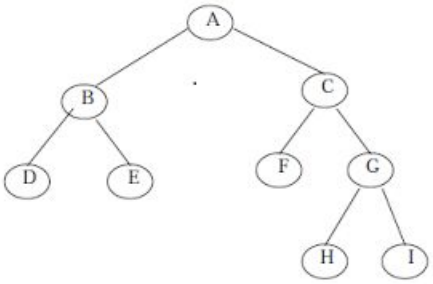
Full Binary Tree



2. **Almost complete binary tree** is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.



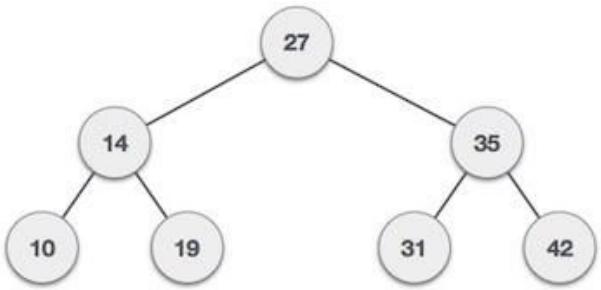
3. **Strictly Binary Tree:** A tree where nodes should have either 0 or 2 child. Not mandatory to put from left to right can be in any order



4. **Binary Search Tree**

A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties −

- The left sub-tree of a node has a key less than or equal to its parent node's key.
- The right sub-tree of a node has a key greater than or equal to its parent node's key.



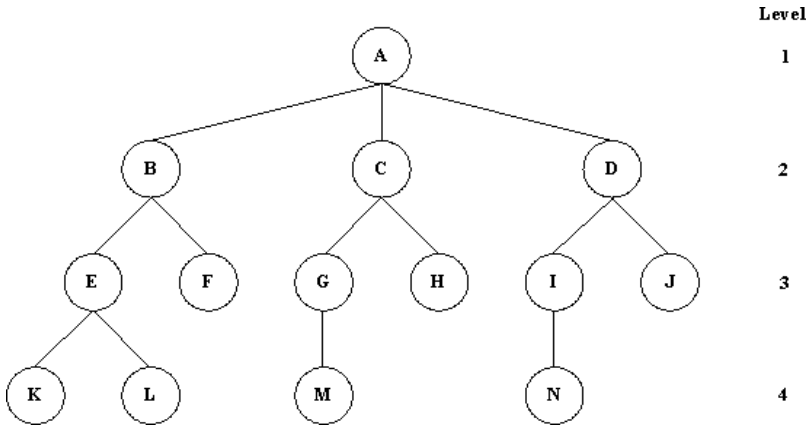
The representation of node in the tree structure:

data	
left child	right child

Define Structure Node for the tree:

```
typedef struct node  *ptr_node;
sturct  node
{
        int   data;
        sturct  node  *left;
        sturct  node  *right;
};
```

General Tree Graphical Picture:

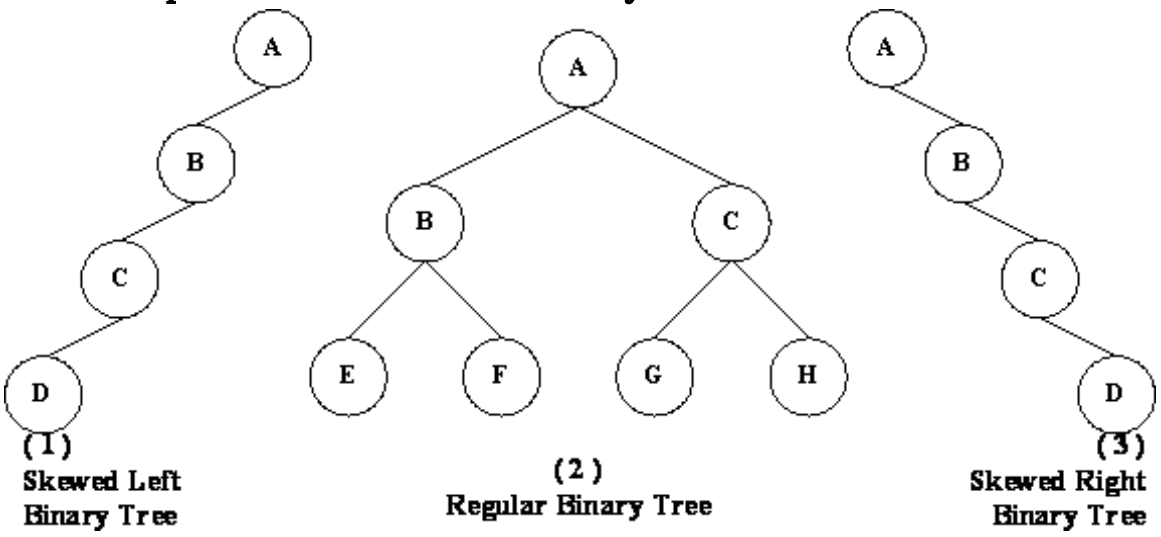


5. Binary Tree

Definition: A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called the left subtree and the right subtree.

Binary Tree Types:
Regular Binary Tree (2)
Skewed Left Binary Tree (1)
Skewed Right Binary Tree (3)

Three Graphical Pictures of the Binary Tree:



6. Properties of Binary Trees

In particular, we want to find out the maximum number of nodes in a binary tree of depth k, and the number of leaf nodes and the number of nodes of degree two in a binary tree. We present both these observations as lemma.

Lemma 5.1 [Maximum number of nodes]:

(1) The maximum number of nodes on level i of a binary tree is 2^{i-1}, $i \geq 1$

(2) The maximum number of nodes in a binary tree of depth k is $2^k - 1$, $k \geq 1$

Proof:
(i) The proof is by induction on i.
Induction Base: The root is the only node on level $i = 1$. Hence the maximum number of nodes on level

$i = 1$ is $2 = 2^{i-1}$.

Induction Hypothesis: For all j, $1 \ j < i$, the maximum number of nodes on level j is 2^{j-1}.

Induction Step: The maximum number of nodes on level $i - 1$ is $2i^{-2}$, by the induction hypothesis. Since each node in a binary tree has maximum degree 2, the maximum number of nodes on level i is 2 times the maximum number on level *i-1* or 2^{i-1}.

(ii) The maximum number of nodes in a binary tree of depth k is (maximum number of nodes on level i. Next, let us examine the relationship between the number of terminal nodes and the number of nodes of degree 2 in a binary tree.

Lemma 5.2 [Relation between number of leaf nodes and nodes of degree 2]: Nonempty binary tree, T, if n_0 is the number of leaf nodes and n_2 the number of nodes of degree 2, then $n_{0 =} n_{2 +} 1$.

Proof: Let $n1$ be the number of nodes of degree 1 and n the total number of nodes. Since all nodes in T are of degree 2 we have:

$n = n_0 + n_1 + n_2$ (1)

If we count the number of branches in a binary tree, we see that every node except for the root has a branch leading into it. If B is the number of branches, then n = B + 1. All branches emanate either from a node of degree one or from a node of degree 2. Thus, B = $n1 + 2n2$. Hence, we obtain

$n = 1 + n_1 + 2n_2$ (2)

Subtracting (2) from (1) and rearranging terms we get

$n0 = n2 + 1$

7. Binary Tree Representations

Array Representation

The numbering scheme used in it suggests out first representation of a binary tree in memory. Since the nodes are numbered from 1 to n, we can use a one-dimensional array to store the nodes. (We do not use the 0^{th} position of the array.) Using Lemma 5.1 we can easily determine the locations of the parent, left child, and right child of any node, i, in the binary tree.

Lemma 5.3:

If a complete binary tree with n nodes (depth = $\lfloor \log_2 n + 1 \rfloor$) is represented sequentially, then for any node with index i, $1 \le i \le n$, we have:

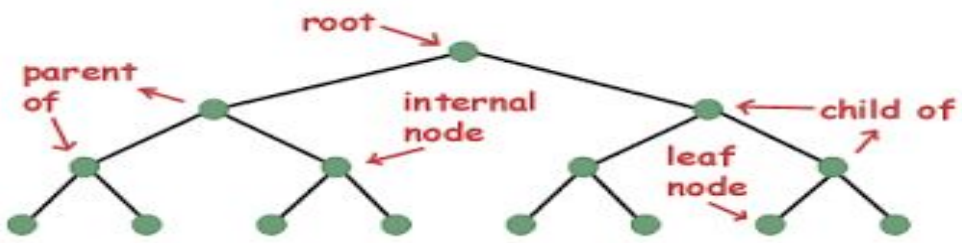
(1) *parent* (i) is at $\lfloor i / 2 \rfloor$ if $i \neq 1$ if $i = 1$, i is at the root and has no parent

(2) *left_child*(*i*) is at $2i$ if $2i \leq n$. If $2i > n$, then *i* has no left child

(3) *rightt_child*(*i*) is at $2i + 1$ if $2i + 1 \leq n$. If $2i + 1 > n$, then *i* has no right child

8. Linked Representation

While the sequential representation is acceptable for complete binary trees, it wastes space for many other binary trees. In, addition, this representation suffers from the general inadequacies of other sequential representations. Thus, insertion or deletion of nodes from the middle of a tree requires the movement of potentially many nodes to reflect the change in the level of these nodes. We can easily overcome these problems by using a linked representation. Each node has three fields, left_child, data, and right_child as two pictures show the node representation of the binary tree below:



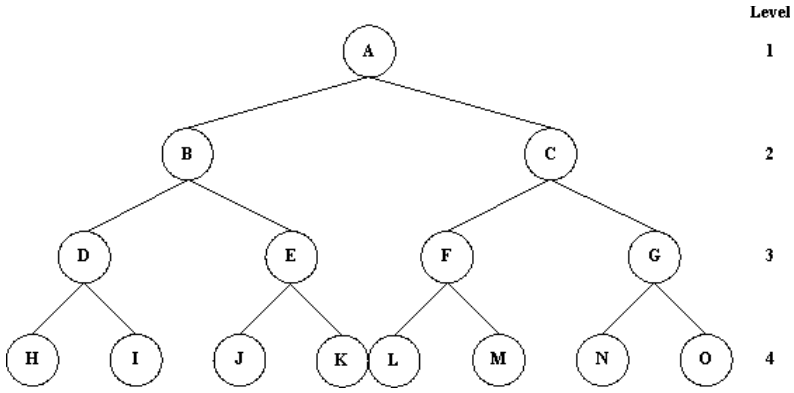
Binary Tree Traversals
There are many operations that we can perform on tree, but one that arises frequently is traversing a tree, that is, visiting each node in the tree exactly once. A full traversal produces a linear order for the information in a tree.

9. Binary Tree Traversals Types

Inorder Traversal (Left, Parent, Right)
Preorder Traversal (Parent, Left, Right)
Postorder Traversal (Left, Right, Parent)

Level Order Traversal (Top to Bottom, Left to Right)

Example of the Binary Tree:



10. Binary Tree Traversals Functions

10.1. Inorder Tree Traversal

Recursive function:

```
void inorder(NODE root)
{
   if(root!=NULL)
   {
          inorder(root->left);
          printf("%d ",root->info);
          inorder(root->right);
       }
}
```

Result of binary tree example:
H, D, I, B, J, E, K, A, L, F, M, C, N, G, O

10.2. Preorder Tree Traversal

Recursive function:

```
void preorder(NODE root)
{
       if(root!=NULL)
       {   printf("%d ",root->info);
           preorder(root->left);
           preorder(root->right);
       }
}
```

Result of binary tree example:
A, B, D, H, I, E, J, K, C, F, L, M, G, N, O

10.3. Postorder Tree Traversal

Recursive function:

```
void postorder(NODE root)
{       if(root!=NULL)
        {    postorder(root->left);
             postorder(root->right);
             printf("%d ",root->info);
        }
}
```

Result of binary tree example:
H, I, D, J, K, E, B, L, M, F, N, O, G, C, A

10.4. Level Order Traversal (Top to Bottom, Left to Right)

```
void level_order(NODE  ptr)
{
        int   front=0, rear = 0;
        NODE  queue[20];
        if(!ptr)            // empty tree;
                return;
        queue[rear++]=ptr;
        //addq(front, &rear, ptr);
        while(front<rear)
        {       ptr = queue[front++];
                if(ptr!=NULL)
                {
                        printf("\n %d ",ptr->info);
                        if (ptr->left)
                                queue[rear++]=ptr->left;
                        if (ptr->right)
                                queue[rear++]=ptr->right;
                }
         printf(" front=%d rear=%d\n", front, rear);
        }
        printf("  \n ending.... \n\n");
}
```

11. Binary Search Tree

Definition: A binary search tree is a binary tree. It may be empty. If it is not empty, it satisfies the following properties:

(1) Every element has a key, and no two elements have the same key, that is, the keys are unique.
(2) The keys in a nonempty left subtree must be smaller than the key in the root of the subtree.
(3) The keys in a nonempty right subtree must be larger than the key in the root of the subtree.
(4) The left and right subtrees are also binary search trees.

12. Inserting Into A Binary Search Tree

To insert a new element, key, we must first verify that the key is different from those of existing elements. To do this we search the tree. If the search is unsuccessful, then we insert the element at the point the search terminated.

Node Defintion

```
struct node
{
   int info;
   struct node *left,*right;
};
typedef struct node *NODE;
```

Tree Creation – Function to insert into tree

```
NODE  insert(NODE root,int data)
{
        NODE newnode, parent, cur;
        newnode=(NODE)malloc(sizeof(struct node));
        newnode->left=newnode->right=NULL;
        newnode->info=data;
        if(root==NULL)
        {
                root=newnode; return root;
        }
        parent=NULL; cur=root;
        while(cur!=NULL)
        {
                if(cur->info==data)
                {
                        printf("\nNode is already present in the tree\n");
                        return(root);
```

```
                }
                if(cur->info < data)
                {       parent=cur; cur=cur->right;  }
                else
                {       parent=cur; cur=cur->left;  }
        }
        if(parent->info<data)
                parent->right=newnode;
        else
           parent->left=newnode;
        return(root);
}
```

13. Searching A Binary Search Tree

Suppose we wish to search for an element with a key. We begin at the root. If the root is NULL, the search tree contains no elements and the search is unsuccessful. Otherwise, we compare key with the key value in root. If key equals root's key value, then the search terminates successfully. If key is less than root's key value, then no elements in the right subtree subtree can have a key value equal to key. Therefore, we search the left subtree of root. If key is larger than root's key value, we search the right subtree of root.

Recursive Function for Binary Search Tree:

```
NODE*   search (NODE*   root, int   key )
{
        if ( root==NULL )
                return   NULL;
        if ( key = = root->data )
                return   root;
        if ( key < root->data )
                return   search ( root->left_child, key );
        return   search ( root->right_child, key );
}
```

Iterative Search Function for Binary Search Tree

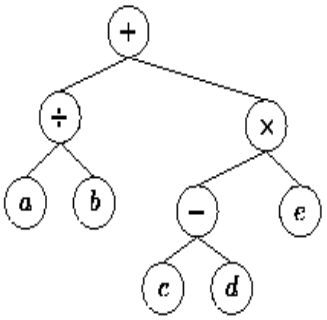
```
void  search_node(NODE root, int key)
{
          NODE cur,parent,successor;
          if(root==NULL)
          {
                printf("\nTree is empty\n");
                return ;
           }
           cur=root;
```

```
        while(cur!=NULL)
        {
            if(key==cur->info)
               break;
            parent=cur;
            if(key < cur->info)
                    cur=cur->left;
            else
                    cur=cur->right;
        }
       if(cur==NULL)
       {
            printf("\nData is not found\n");
            return ;
       }
       printf("\nData %d is found\n",key);
       return  ;
}

int main()
{
   NODE root=NULL;
   NODE create(NODE,int);
   NODE delete_node(NODE,int);
   //clrscr();
   while(1)
   {
            printf("1:Create Tree\n");
            printf("2:TREE TRAVERSAL\n");
            printf("3.SEARCH\n");
            printf("4.Level_Order\n");
            printf("5.EXIT\n");
            printf("\nEnter your choice=\n");
            scanf("%d",&choice);
            switch(choice)
            {
          case 1    :     printf("\nEnter data to be inserted\n");
                          scanf("%d",&data);
                          root=create(root,data);
                          break;
          case 2    :     if(root==NULL)
                                printf("\nEMPTY TREE\n");
                          else
                          {     printf("\nThe inorder display : ");
                                inorder(root);
                                printf("\nThe preorder display : ");
```

```
                                preorder(root);
                                printf("\nThe postorder display : ");
                                postorder(root);
                                printf("\n");
                        }
                        break;
        case 3 :        printf("\nenter the key to search and delete:\n");
                        scanf("%d",&key);
                        search_node(root,key);
                        break;
        case 4          :       level_order(root);  break;
        case 5 :   return 0;

         }
   }
   return 0;
}
```

14. Expression Trees

a/b+(c-d)*e

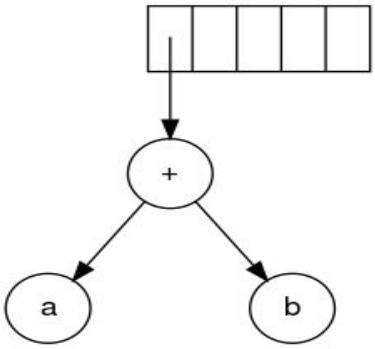


CONSTRUCTION OF AN EXPRESSION TREE

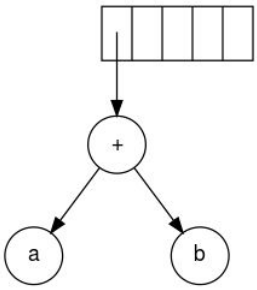
The evaluation of the tree takes place by reading the postfix expression one symbol at a time. If the symbol is an operand, one-node tree is created and a pointer is pushed onto a stack. If the symbol is an operator, the pointers are popped to two trees *T2* and *T1* from the stack and a new tree whose root is the operator and whose left and right children point to *T1* and *T2* respectively is formed . A pointer to this new tree is then pushed to the Stack.

Example

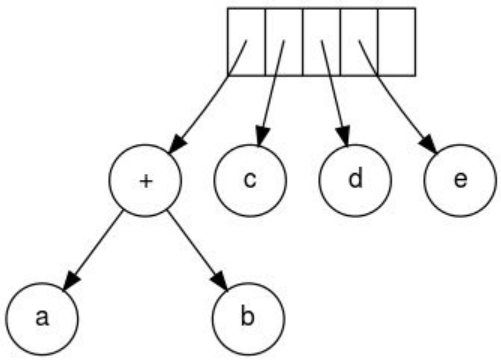
The input is: a b + c d e + * *. Since the first two symbols are operands, one-node trees are created and pointers are pushed to them onto a stack. For convenience the stack will grow from left.



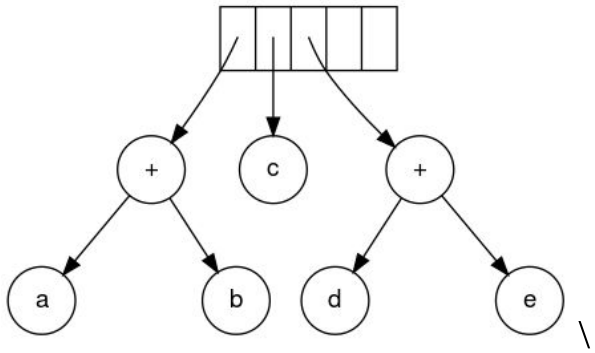
The next symbol is a '+'. It pops the two pointers to the trees, a new tree is formed, and a pointer to it is pushed onto to the stack.



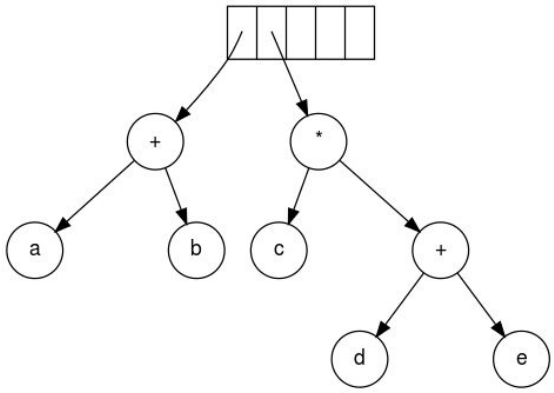
Next, c, d, and e are read. A one-node tree is created for each and a pointer to the corresponding tree is pushed onto the stack.



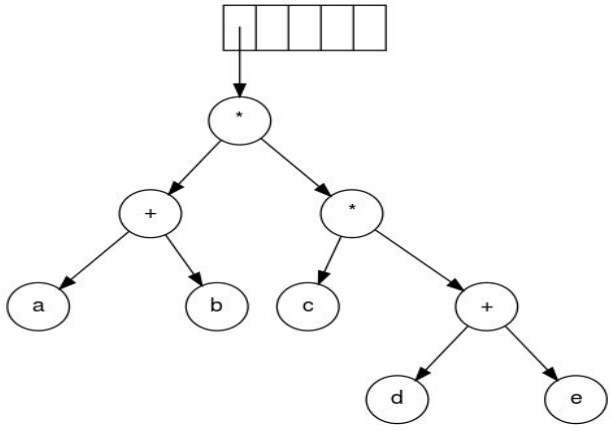
Continuing, a '+' is read, and it merges the last two trees.

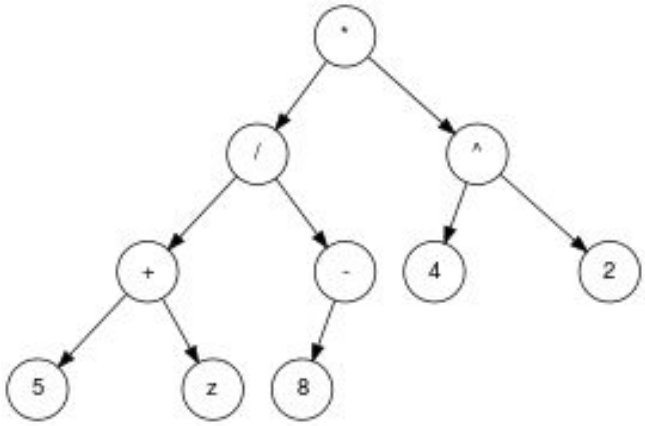


Now, a '*' is read. The last two tree pointers are popped and a new tree is formed with a '*' as the root.

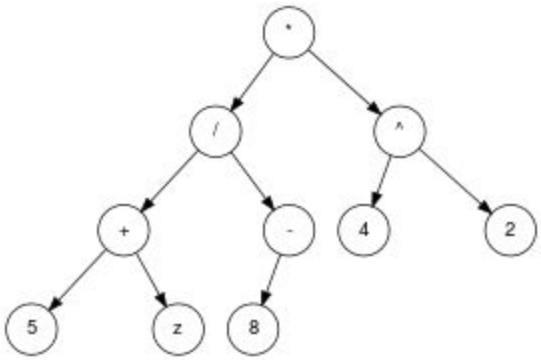


Finally, the last symbol is read. The two trees are merged and a pointer to the final tree remains on the stack.





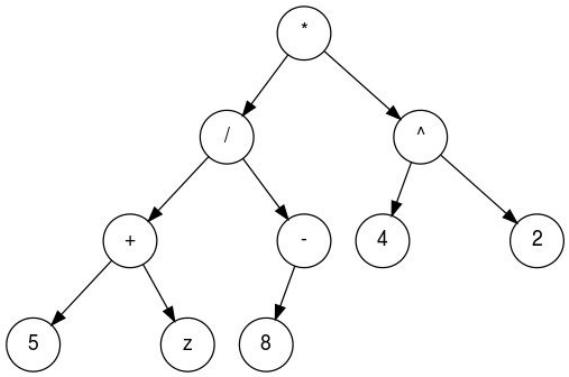
15. Algebraic expressions



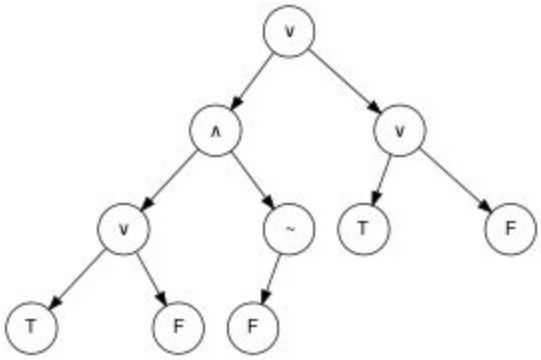
Binary algebraic expression tree equivalent to ((5 + z) / -8) * (4 ^ 2)

Algebraic expression trees represent expressions that contain numbers, variables, and unary and binary operators. Some of the common operators are × (multiplication), ÷ (division), + (addition), − (subtraction), ^ (exponentiation), and - (negation). The operators are contained in the internal nodes of the tree, with the numbers and variables in the leaf nodes.The nodes of binary operators have two child nodes, and the unary operators have one child node.

Binary algebraic expression tree equivalent to ((5 + z) / -8) * (4 ^ 2)

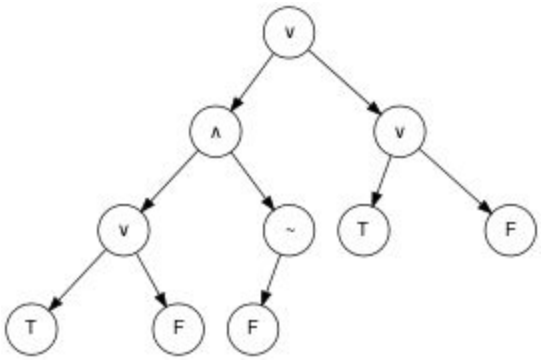


Boolean expressions

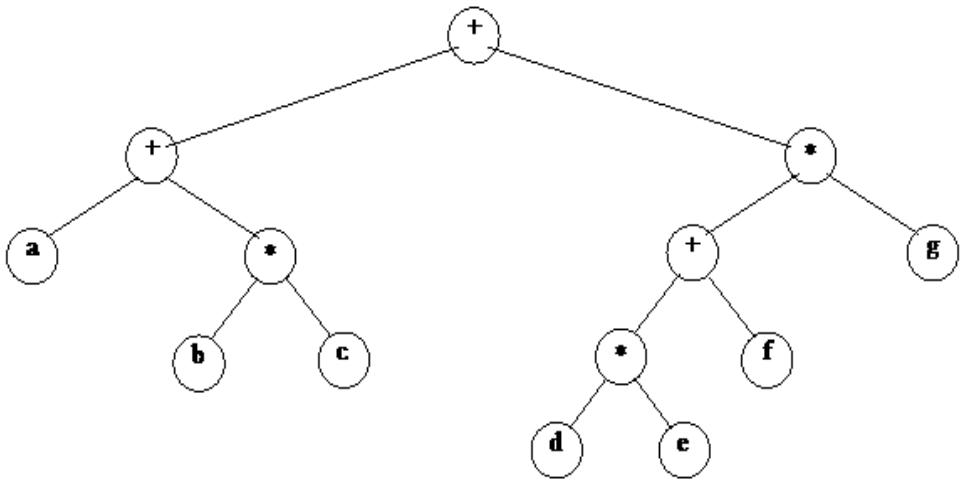


Boolean expressions are represented very similarly to algebraic expressions, the only difference being the specific values and operators used. Boolean expressions use *true* and *false* as constant values, and the operators include (AND) &).

((TorF) and (~F))or(TorF)



Example for Expression Tree;
(a+(b*c))+(((d*e)+f*g)



Inorder: *(a + (b * c)) + (((d * e) + f) * g)*

16. Write a Program to Create a tree for Postfix Expression and Evaluate the Expression using Binary Tree.

Also print the expression in infix, postfix, prefix form using inorder, postorder and preorder respectively.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

struct root
{
        char data;
        struct root *left;
        struct root *right;
};
typedef struct root NODE;
NODE *stack[30];
int top=-1;
int c=0, leaf=0, InternalNodes=0;
NODE* insert(char b)
{
        NODE *temp;
        temp=(NODE*)malloc(sizeof(NODE));
        temp->data=b;
        temp->left=NULL;
        temp->right=NULL;  c++;  // count no of nodes in a tree
        return(temp);
}

void inorder(NODE *t)
{
        if(t!=NULL)
        {
                inorder(t->left);
                printf("%c",t->data);
                inorder(t->right);
        }
}
void preorder(NODE *t)
{
        if(t!=NULL)
        {
                printf("%c",t->data);
```

```
                preorder(t->left);
                inorder(t->right);
        }
}
void postorder(NODE *t)
{
        if(t!=NULL)
        {
                postorder(t->left);
                postorder(t->right);
                printf("%c",t->data);
        }
}

int calculate(int lval, char data, int rval)
{   switch(data)
        {
                case '+' : return (lval + rval);
                case '-' : return (lval - rval);
                case '*' : return (lval * rval);
                case '/' : return (lval / rval);
                default :  printf(" unknown operator"); return -1;
        }
}
```

This function will take single digit numeric character and displays infix, postfix, prefix and evaluates the expression by taking numeric value.

```
int evaluate(NODE *root)
{      int lval, rval, res;
       if (root->left==NULL && root->right==NULL)
        {          return (root->data-'0');
        }
        else
        {         lval=evaluate(root->left);
                          rval=evaluate(root->right);
                  res=calculate(lval, root->data, rval);
                          printf("res=%d\n",res);
                  return res;
       }

}
```

OR

This function will take single alphabetic character and displays infix, postfix, prefix and evaluates expression the by taking the input from the keyboard.

```
int evaluate(NODE *root)
{       int lval, rval, res, val;
        if (root->left==NULL && root->right==NULL)
        {
                        if(root->data>='0' && root->data<='9')
                                return (root->data-48);
                else
                {
                        printf("enter the value for variable %c =", root->data);
                        scanf("%d",&val);
                        return val;
                }
        }
        else
        {         lval=evaluate(root->left);
                          rval=evaluate(root->right);
                          res=calculate(lval, root->data, rval);
                          printf("res=%d\n",res);   return res;
        }
 }

int main()
{
        char a[20]; NODE* temp, t;  nt j, i, res;
        printf("\nEnter the postfix expression");
        gets(a);
        for(i=0;a[i]!='\0';i++)
        {
                if(a[i]=='*' || a[i]=='/' || a[i]=='+' || a[i]=='-')
                {
                        temp=insert(a[i]);
                        temp->right=stack[top--];
                        temp->left=stack[top--];
                        stack[++top]=temp;
                }
                else
                {
                        temp=insert(a[i]);
                        stack[++top]=temp;
                }
        }
        printf("\n  Inorder Traversal :");
        inorder(temp);
        printf("\n");
        printf("\n  Preorder Traversal :");
        preorder(temp);
        printf("\n");
        printf("\n  Postorder Traversal :");
```

```
        postorder(temp);
        res=evaluate(temp);
        printf("\nResult of Expression=%d\n",res);
                printf("\n Total no nodes =%d\n",c);
        return 0;
}
```

17. Function for finding Height of a Binary Tree

```
int max(int a, int b)
{     if(a>b) return a;     else     return b;
}
int height(NODE *root)
{               if(root==NULL)
                        return -1;
                else
                        return max(height(root->left), height(root->right))+1;
}
}
```

18. Function for counting the number of leaf nodes and internal nodes

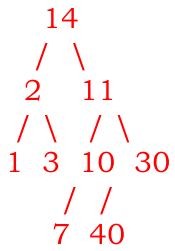
```
    Int leaf=0, InternalNodes=0;

void  Leaf_NonLeafNodes(NODE *root)
{   if (root->left==NULL && root->right==NULL)
    {       leaf++;  //increment  leaf nodes
    }
    else
    {           InternalNodes++;  // Increment  internal nodes count
                Leaf_NonLeafNodes(root->left);
                        Leaf_NonLeafNodes(root->right);
    }
            return ;
}
```

19. Questions on Tree

1. Define Tree, binary tree, root, leaf, parent, sibling, ancestor, successor, path, degree, depth, level.
2. Define Internal nodes, External nodes
3. Mention different types of trees..
4. What is Binary search tree(BST)? What are its properties?
5. Explain how tree can be represented with example.
6. Discuss tree traversal methods

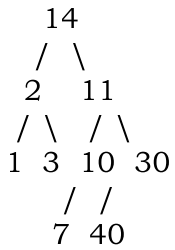
7. Write a function for tree traversal.
8. Write a function to search a node in a BST.
9. Write a function to insert into BST.
10. Write functions to find a height of tree, total number of nodes, total number of leaf nodes and internal nodes.
11. Write a c function to construct binary for the postfix expression and evaluate the expression.
12. Construct binary search tree for the following data
    - 12.1. 14, 15, 4, 9, 7, 18, 3, 5, 16, 4, 20, 17.14, 9(indicate msg for duplicate nos.)
    - 12.2. D, A, T, S, R, U, E, B, N, F, W, V, K, P
    - 12.3. 100, 23, 79, 43, 200, 230, 142, 32, 145,44, 56, 134, 220
13. Construct expression tree for the arithmetic expressions. Write prefix and postfix expressions
    - 13.1. ASB*C-D+E/F(G+H)
    - 13.2. A-B/(C*D^E)
    - 13.3. (A+B)+C/D
    - 13.4. A*(B+C)*D
    - 13.5. (((A/B)-C)+(D+E)/((F+A*D)+C)
    - 13.6. (A+B)*D+E/(F+A*D)+C
    - 13.7. A&&B ||c||!(E>F)
    - 13.8. ((A+B) > C)||((B+C) > A) || ((C+A)>B)
    - 13.9. ((B*B+C*C)==A*A) || ((C*C+A*A)== B*B)||((A*A+B*B==C*C)
    - 13.10. (A==B|| B==C || C==A)

Multiple Choice



1. There is a tree in the box at the top of this section. How many leaves does it have?
   - A. 2 B. 4 C. 6 D. 8 E. 9
2. There is a tree in the box at the top of this section. How many of the nodes have at least one sibling?
   - A. 5 B. 6 C. 7 D. 8 E. 9
3. There is a tree in the box at the top of this section. What is the value stored in the parent node of the node containing 30?
   - A. 10 B. 11 C. 14 D. 40
   - E. None of the above
4. There is a tree in the box at the top of this section. How many descendants does the root have?
   - A. 0 B. 2 C. 4 D. 8

5. There is a tree in the box at the top of this section. What is the depth of the tree?
   - A. 2 B. 3 C. 4 D. 8 E. 9
6. There is a tree in the box at the top of this section. How many children does the root have?
   - A. 2 B. 4 C. 6 D. 8 E. 9
7. Consider the binary tree in the box at the top of this section. Which statement is correct?
   - A. The tree is neither complete nor full.
   - B. The tree is complete but not full.
   - C. The tree is full but not complete.
   - D. The tree is both full and complete.
8. What is the minimum number of nodes in a full binary tree with depth 3?
   - A. 3 B. 4 C. 8 D. 11 E. 15
9. What is the minimum number of nodes in a complete binary tree with depth 3?
   - A. 3 B. 4 C. 8 D. 11 E. 15
10. Select the one true statement.
   - A. Every binary tree is either complete or full.
   - B. Every complete binary tree is also a full binary tree.
   - C. Every full binary tree is also a complete binary tree.
   - D. No binary tree is both complete and full.
11. Suppose T is a binary tree with 14 nodes. What is the minimum possible depth of T?
   - A. 0 B. 3 C. 4 D. 5
12. Select the one FALSE statement about binary trees:
   - A. Every binary tree has at least one node.
   - B. Every non-empty tree has exactly one root node.
   - C. Every node has at most two children.
   - D. Every non-root node has exactly one parent.
13. Consider the binary tree node. Which expression indicates that t represents an empty tree?
   - A. (t == NULL)
   - B. (t->data( ) == 0)
   - C. (t->data( ) == NULL)
   - D. ((t->left( ) == NULL) && (t->right( ) == NULL))
14. Consider the node of a complete binary tree whose value is stored in data[i] for an array implementation. If this node has a right child, where will the right child's value be stored?
   - A. data[i+1] B. data[i+2]
   - C. data[2*i + 1] D. data[2*i + 2]
15. How many recursive calls usually occur in the implementation of the tree_clear function for a binary tree?
   - A. 0 B. 1 C. 2
16. Suppose that a binary taxonomy tree includes 8 animals. What is the minimum number of NONLEAF nodes in the tree?
   - A. 1 B. 3 C. 5 D. 7 E. 8



17. There is a tree in the box at the top of this section. What is the order of nodes visited using a pre-order traversal?
    - A. 1 2 3 7 10 11 14 30 40
    - B. 1 2 3 14 7 10 11 40 30
    - C. 1 3 2 7 10 40 30 11 14
    - D. 14 2 1 3 11 10 7 30 40
18. There is a tree in the box at the top of this section. What is the order of nodes visited using an in-order traversal?
    - A. 1 2 3 7 10 11 14 30 40 (B). 1 2 3 14 7 10 11 40 30
    - C. 1 3 2 7 10 40 30 11 14 (D). 14 2 1 3 11 10 7 30 40
19. There is a tree in the box at the top of this section. What is the order of nodes visited using a post-order traversal?
    - A. 1 2 3 7 10 11 14 30 40
    - B. 1 2 3 14 7 10 11 40 30
    - C. 1 3 2 7 10 40 30 11 14
    - D. 14 2 1 3 11 10 7 30 40