UNIT-V

Tree : Definition of tree, Fundamental terminologies-Node, Child, Parent, Root, Leaf, Level, Height and Subling. Rooted trees, Ordered trees, Binary tree, Complete binary tree, Tree of an algebraic expression, Tree searching (traversal algorithms)- Preorder, Inorder and Postorder. Distance and centre, Relation between general tree and binary tree, Spanning trees, Algorithms for minimal spanning trees (Kruskal's and Prim's). Game tree.

9 TREE

Tree

A tree is a connected graph without any cycle. For example, the graphs in fig.1 are trees with one, two, three, four vertices. A tree must be a simple graph because self-loops and parallel edges, both form cycles. A graph is called acyclic if it contains no cycles. Thus a tree is a connected acyclic graph. A collection of trees is called a forest.

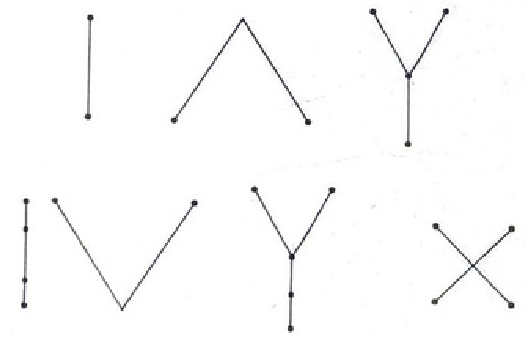


Fig. 1

Note : A tree is called trivial if it has only one vertex.

Theorem : A tree with n vertices has exactly (n-1) edges.

Proof We shall prove the statement by induction on the number of vertices n. A tree with one vertex is the trivial tree, which has zero edges. So it is true for n = 1. As an induction hypothesis, assume for some number k>1, that every tree with K has exactly (k-1) edges. Now consider any

tree with (k+1) vertices, then by the above theorem T contains a vertex of degree one say v. then the graph T - v is acyclic since deleting v from an acyclic graph cannot create a cycle. Moreover, T - v is connected, since the vertex v has degree one in T. thus, T-v is a tree with K vertices, and hence it has exactly (K-1) edges by introduction assumption. But, T-v is obtained by deleting the vertex v of degree one, so T-v has one edge fewer than T. hence T has K edges. This completes the proof.

Theorem 2 : a graph G is a tree if it is minimally connected

Proof A connected graph is said to be minimally connected if removal of any one edge from it disconnects the graph.

Let G be a tree and suppose that it is not minimally connected. Then there must be an edge, say e in G such that G – e is connected. It implies that e must be in some cycle in G, contradicting the fact that G is a tree without any cycle. Hence the tree G must be minimally connected.

Conversely, suppose that G is a minimally connected graph. Then it cannot have a cycle, otherwise by deleting one of the edges in the cycle, we obtain a graph still connected thus G is a tree.

Distance between two vertices in a graph

In a connected graph G the distance d (u, v) between two of the distinct two of its distinct vertices u and v, is the length of the shortest u – v path.

For example, some of the u – v paths between the vertices u and v in the following graph are $ucu_4fu_5hu, ucu_4eu_3gu_5hu, ucu_4fu_5ju_6lu, ucu_4eu_3gu_5ju_6lu, udu_2du_3eu_4fu_5hu$. But the shortest path is ucu_4fu_5hu which is of length 3. Hence d (u,v) = 3

In a tree, there is precisely one path between any pair of vertices, so it becomes much easier to determine the distance between any two of its vertices.

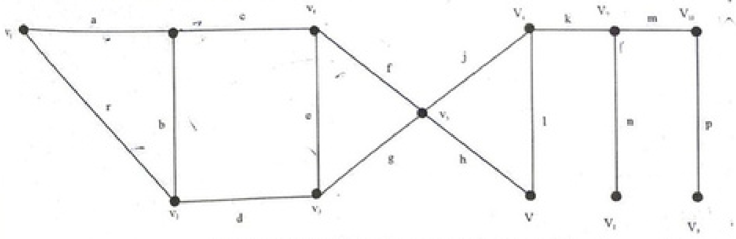


Fig. 2 : Distance between u and v is three

Eccentricity of vertex

The eccentricity E (v) of a vertex v in a graph G = (V, E) is the distance of the farthest vertex v, from v in G. thus

$$E(v) = \max\{d(v, v_i) \mid v_i \in v\}$$

Center of a Graph

The vertex in the graph G having minimum eccentricity is called the center of G. A graph may have centers for example, in a cycle graph every vertex is a center. However, a tree has one or two centers only.

Theorem 3: Every tree has either one or two centers.

Proof. Let T be a tree. If T has only one vertex, then it is the center of T. if T has two vertices, then both are the centers of T. now suppose that T has more than two vertices. Since the maximum distance from a given vertex v to any other vertex v_i occurs only when v_i is a pendant vertex and every tree has at least two pendant vertices, therefore, if we delete all the pendant vertices from the tree T, the resulting graph T1 is still a tree in which the eccentricities of all the vertices are reduced by one. Therefore, all the vertices that are the centers of T, still remain the centers in T1. Again we delete all the pendent vertices in t1 to get another tree T2 that has same centers as that of T1. We continue the process of deletion vertices until we are left with either one vertex (which is the center of T) or an edge (whose end vertices are the centers of T). This completes the proof.

The deletion process of pendent vertices in a tree T is shown in the following figure. The numbers associated with the vertices denote their eccentricities.

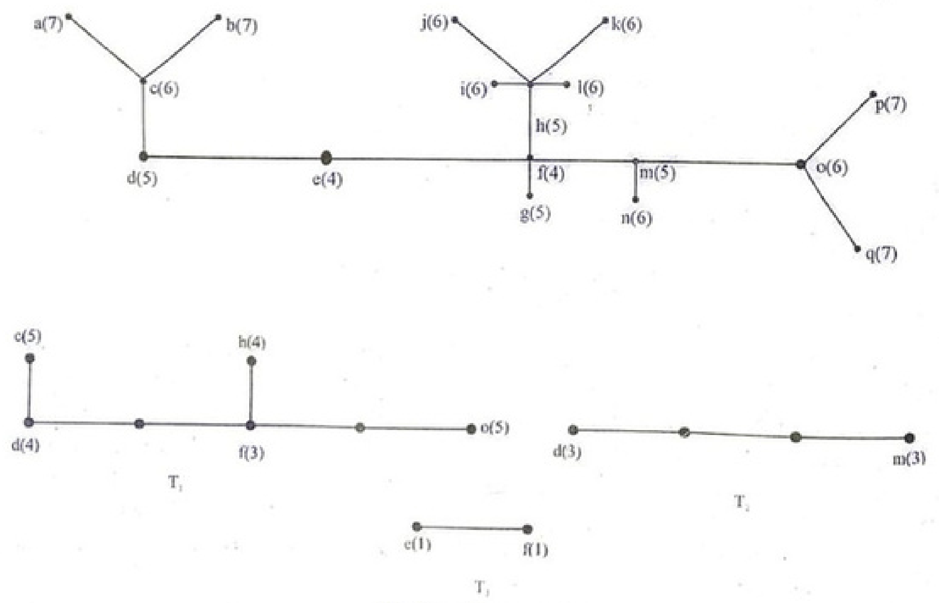


Fig.3 : Tree centers

Radius and Diameter of a Tree

In a tree, the eccentricity of the center (or center) (i.e., the distance from the center of the tree to the farthest vertex) is said to be the radius of the tree.

The diameter of a tree T is defines as the length of the longest path in T.

The radius of the tree T in Fig.3 is thus 4 and the diameter in 7.

Rooted Tree

In many trees a particular vertex of the tree is designated as the root. Once a root of a tree is specified, the direction of each edge can be assigned. Since there is only one path from the root to each vertex of the tree, we direct each vertex away from the root. Thus a tree together with its root intends a directed graph called the rooted tree.

Thus in a rooted tree one particular vertex called the root is distinguished from all the other vertices. To differentiate the rooted tree from no rooted trees, the root of the tree is marked distinctly by enclosing a small triangle.

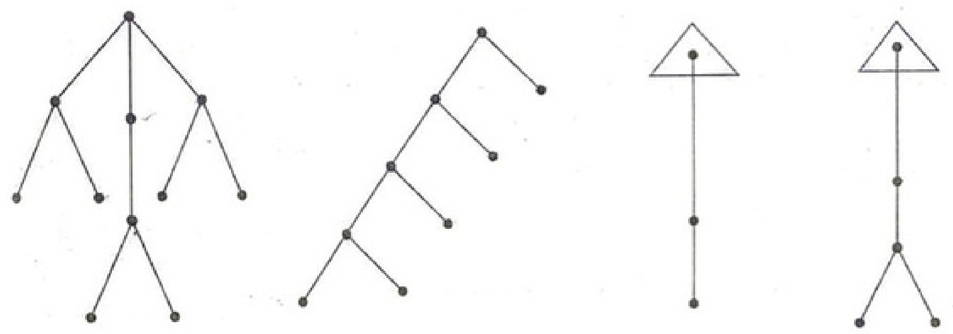


Fig.4 : Rooted trees

The terminology often used with rooted tree has botanical and genealogical origins. Suppose that T is a rooted tree. If a vertex u of T other than the root is adjacent to the vertex v and v lies in the level below u , then v is called a child of u and u is called the parent of v. vertices having the same parent are called siblings. The ancestors of a vertex other than the root are the vertices in the path form the root to this vertex. All these vertices that have a vertex v as an ancestor are called descendants of v. a vertex in a rooted tree without any child is called a leaf. All these vertices that have children are called the internal vertices. The root of a rooted tree T is an internal vertex unless it is the only vertex in T, in which case it is a leaf.

A rooted tree T is said to be an m-ary tree if every internal vertex of T has at most m children.

A tree T is said to be a complete m-ary tree if every vertex in T has exactly m children or no children.

Ordered Trees

If in a tree at each level, an ordering is defined, then such a tree is called an ordered tree. e.g., the trees shown in Figs. 5 and 6 represent the same tree but have different orders.

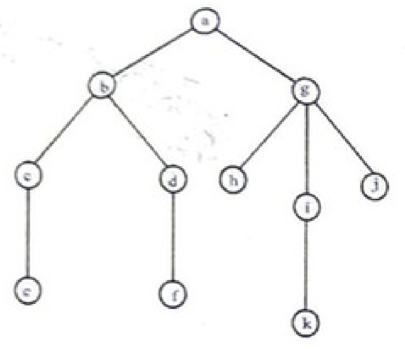


Fig.5

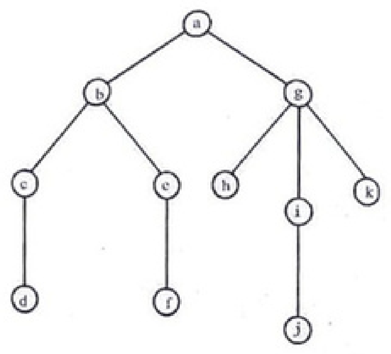


Fig.6

Path Length of A Vertex

The path length of a vertex in a rooted tree is defined to be number of edges in the path from the root to the vertex.

Example: find the path lengths of the nodes b, f, l, q in fig.7

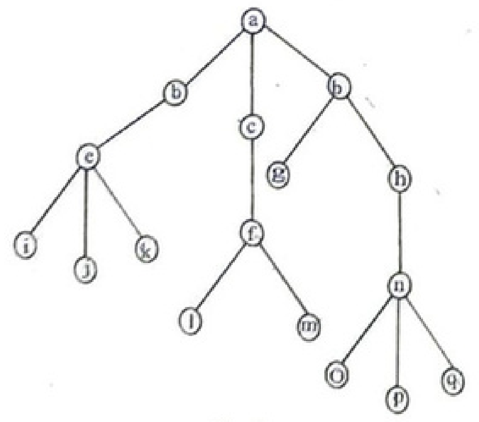


Fig.7

Sol: The path length of node b is one.

The path length of node f is two.

The path length of node l is three.

The path length of node q is four.

Theorem4: prove that there is one path between every pair of vertices in a tree T.

Proof: we know that T is a connected graph, in which there must exist at least one path between every pair of vertices. Now assume that there exists two different paths from some node a to some node b of T, the union of these two paths will contain a cycle and therefore T cannot be a tree. Hence, there is only one path between every pair of vertices in a tree.

Forest

If the root and the corresponding edges connecting the nodes are deleted from a tree. We obtain a set of disjoint trees. This set of disjoint trees is called a forest

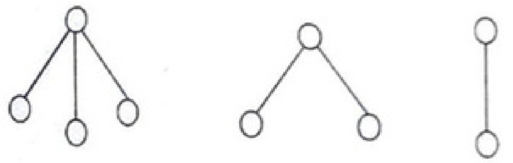


Fig. 8

Binary Tree

If the out degree of every node is less than or equal to 2, in a directed tree then tree is called a binary tree. A tree consisting of no nodes (empty tree) is also a binary tree. A binary tree is shown in figs. 9 and 10.

Basic Terminology

Root. A binary tree has a unique node called the root of the tree.

Left Child. The node to the left of the root is called its left child.

Right Child. The node to the right of the root is called its right child.

Parent. A node having lefty child or right child or both is called parent of the nodes.

Siblings. Two nodes having the same parent are called siblings.

Leaf. A node with no children is called a leaf. The number of leaves in a binary tree can vary from one

(minimum) to half the number of vertices (maximum) in a tree.

Ancestor. If a node is called descendent of another node if it is the child of the node or child of some other descendent of that node. The root is an ancestor of every other node in the tree.

Descendent. A node is called descendent of another node if it is the child of the node or child of some other descendent of that node. All the nodes in the tree are descendents of the root.

Left Sub tree. The sub tree whose root is the left child of some node is called the left sub tree of that mode.

Example: for the tree as shown in fig.9

I. Which node is the root?

II. Which nodes are leaves?

III. Name the parent node each node.

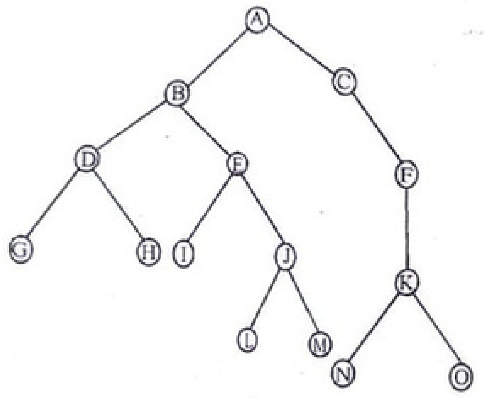


Fig.9

Sol.

I. The node A is the root node.

II. The nodes G, H, I, L, M, N, O are leaves

III.

Nodes	Parent
B, C	A
D, E	B
F	C
G, H	D
I, J	E
K	F
L, M	J
N, O	K

Right Sub tree. The sub tree whose root is the right child of some node is called the right subtree of that node.

Level of a Note. The level of a node is its distance from the root. The level of root is defined as zero. The level of all other nodes is one more than its parent node. The maximum number of nodes at any level N 2n.

Depth or Height of a Tree. The depth or height of a tree is defined as the maximum number of nodes in a branch of root id one. The maximum number of nodes in a binary tree of depth d is 2d - 1 where d > 1.

External Nodes. The nodes which have no children are called external nodes or terminal nodes or non-terminal nodes.

Internal Notes. The nodes which have one or more than one children are called internal nodes or non-terminal nodes.

Example. For the tree as shown in fig.10

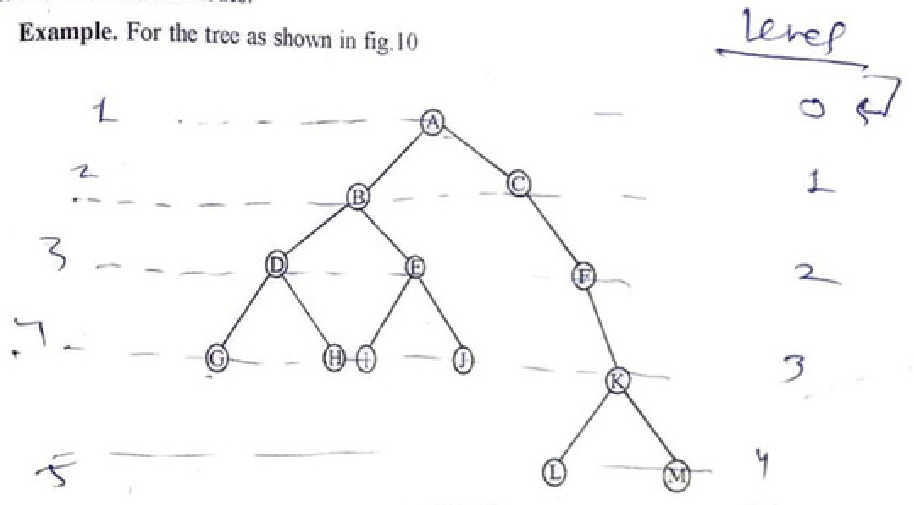


Fig.10

I. List the children of each node.

II. List the siblings

III. Find the depth of each node.

IV. Find the level of each node.

Sol.

1. The children of each node is as follows:

Nodes	Parent
A	B, C
B	D, E
C	F

D	G, H
E	I, J
F	K
K	L, M

2. The siblings are as follows:

Siblings

B and C

D and E

G and H

I and J

L and M are all siblings.

3.

Node	Depth or Height
A	1
B, C	2
D, E, F	3
G, H, I, J, K	4
L, M	5

IV.

Node	Depth or Height
A	0
B, C	1
D, E, F	2
G, H, I, J, K	3
L, M	4

Binary Expression Trees

Algebraic expression can be conveniently expressed by the expression tree. An expression having binary operators can be decomposed into.

< left operand or expression >(operator)< right operand or expressions >

Depending upon precedence of evaluation.

The expressions tree is a binary tree whose root contains the operator and whose left subtree contains the left expressions the left expressions and right subtree contains the right expressions.

Example

Construct the binary expression tree for the expression (a + b) * (d / c)

Sol. The binary expressions tree for the expression (a + b) *

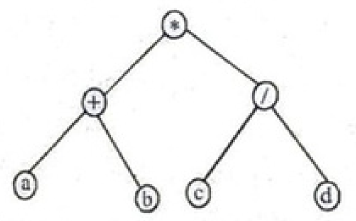


Fig.11

Complete Binary Tree

Complete binary tree is a binary tree if all its levels, expect possibly the last, have maximum number of possible nodes as for left as possible. The depth of complete binary tree having n nodes is log 2 n+1.

For example: the tree shown in fig. 12 is a complete binary tree.

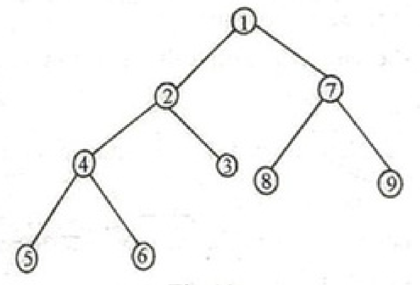


Fig.12

Full Binary Tree

Full binary tree is a binary tree in which all the leaves are in the same level and every non-leaf node has two children.

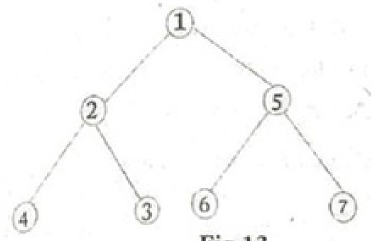


Fig.13

For example: the tree shown in fig.13 is a full binary tree.

Theorem5. Prove that the maximum number of nodes or level n of a binary tree is 2^n, where n>0.

Basic of Induction. The only node at level n = 0 is the root node. Thus, the maximum number of nodes on level n = 0 is $2^0 = 1$

Induction hypothesis. Now assume that it is true for level j, where $n \geq j \geq 0$. Therefore the maximum no. of nodes on level j is 2^j.

Induction Step. By induction hypothesis, the maximum number of nodes on level j − 1 is $2^j - 1$. Since, we know that each node on level j is twice the maximum number of level $j-1$.

Hence, at level j, the maximum number of nodes is

$$= 2.2^{j-1}$$

$$= 2^j \text{. hence proved}$$

Theorem6 Prove that the maximum number of nodes in a binary tree of depth d is $2^d - 1 = 1$

Proof. This is can be proved ny induction.

Basis of Induction. The only node at depth d=1 is the root node. Thus, the maximum number of nodes on depth d = 1 is $2^1 - 1 = 1$.

Induction Hypothesis. Now assume that it is true for depth $k, d > k \geq 1$ Therefore, the maximum number of nodes on depth K is $2^k - 1$

Induction step. By induction hypothesis, the maximum number of nodes on depth K − 1 is $2^{k-1} - 1$. since, we know that each node in a binary tree has maximum degree 2, therefore, the maximum number of nodes on depth K − 1.

So, at depth d = K, the maximum number of nodes is $= \left(2.2^{k-1}\right) - 1$

$$2^{k-1+1} - 1 = 2^k - 1 \text{ Hence proved.}$$

Theorem7. Prove that in a binary tree, if Ne is the number of external nodes or leaves and nl is the no. of internal nodes, then $n_g = n_i + n_0$

Proof. Let n be the total number of nodes in the tree. then we may have three types of nodes in the tree.

n_E = the number of nodes having zero degree.

n_I = the number of nodes having two degree.

n_0 = the number of nodes having one degree.

So, we have $n = n_e + n_I + n_0$

Let us assume that the number of edges of the tree us E. So, with these E edges we can connect E + 1 nodes.

Hence $n = E + 1$

Since, all edges are either from a node of degree one or from a node of degree two,

Therefore, $E = n_0 + 2n_I$

Put this value in the eq. (2), we have

$$n = n_0 + 2n_1 + 1$$

Subtract eq. (4) from eq. (1), we get

$$n_E = n_i + 1$$

Hence proved.

Traversing Binary Trees

Traversing means to visit all the nodes of the tree. There are these standard methods to traverse the binary trees. These are as follows:

1. Preorder traversal
2. In order traversal
3. Post order traversal

1. Preorder traversal. The preorder traversal of a binary tree is recursive process. The preorder traversal traversal of a tree is

i. Visit the root of the tree.

ii. Traverse the left subtree in preorder.

iii. Traverse the right subtree in preorder.

2. Postorder traversal. the postorder traversal of a binary tree is a recursive process. the Postorder traversal of a tree is.

i. Traverse the left subtree in postorder.

ii. Traverse the right subtree in postorder.

iii. Visit the root of the tree.

3. Inorder traversal. The inorder trsaversal of a binary tree is a recursive process. the inorder traversal of a tree is.

i. Traverse in inorder the left subtree.

ii. Visit the root of the tree.

iii. Traverse in inorder the right subtree.

Example Determine the preorder, postorder and inorder traversal of the binary tree as shown in fig. 14

In order

L | Root | Ryt

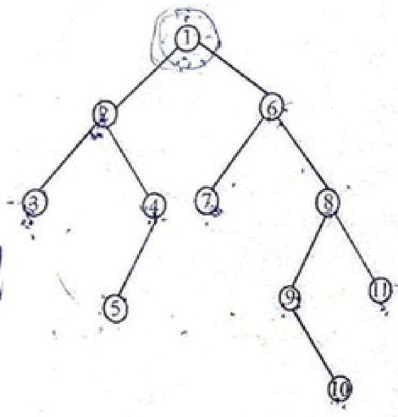


Fig.14

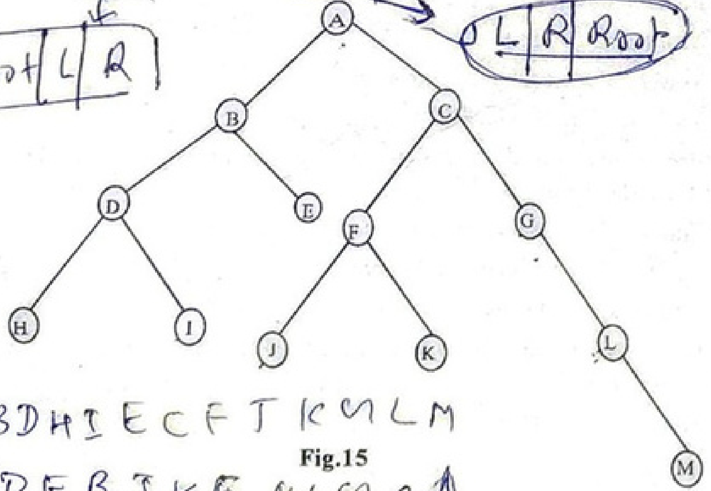
Sol. The preorder; postorder and inorder traversal of the tree is as follows:

Preorder	: 1	2	3	4	5	6	7	8	9	10	11
Postorder	: 3	5	4	2	7	10	9	11	8	6	1
Inorder	: 3	2	5	4	1	7	6	9	10	8	11

Example Give the preorder, inorder and postorder traversals of the tree shown in Fig.15

Root | L | R

L | R | Root



A B D H I E C F J K M L M

Fig.15

H I D E B J K F M L G C A

Sol.

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

ALGOROTHMS

(a) Algorithm to Draw a Unique Binary Tree When Inorder and Preorder Traversal of the Tree is Given.

1. We know the root of the binary tree is the first node in its preorder. Draw the root of the tree.
2. To find the left child of the root node, first use the inorder traversal to find the nodes in the left subtree of the binary tree. (all the nodes that are left to the root node in the inorder traversal are the nodes of the left subtree). After that the left child of the root is obtained by selecting the first node in the preorder traversal of the left subtree. Draw the left child.
3. In the same way, use the inorder traversal to find the nodes in the nodes in the right subtree of the binary tree. then the right child is obtained by selecting the first node in the preorder traversal of the right subtree. Draw the right child.
4. Repeat the steps 2 And 3 with each new node until every node is not visited in preorder. Finally, we obtain the unique tree.

Example Draw the unique binary tree when the inorder and preorder traversal is given as follows:

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

Sol. We know that the root of the binary tree is the first node in a preorder traversal. now check A, in the inorder traversal , all the nodes that are left to A, are nodes of left subtree and all the nodes that are right of A, are nodes of right subtree. Read the next node in preorder and check its position against the root node, if it is left of root node, then draw it as left child, otherwise draw it as right child. Repeat the above process for each new node until all the nodes of preorder traversal are read and finally we obtain the binary tree as shown in fig. 16

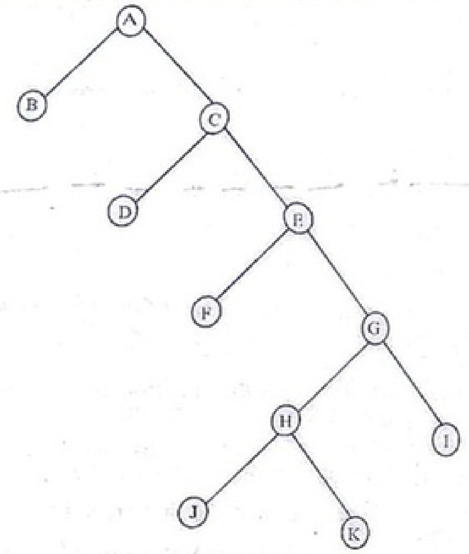


Fig .16

Example Draw the unique binary tree when the following is given:

Inorder	:	d	b	h	e	a	f	c	i	j	g
Preorder	:	a	b	d	e	h	c	f	g	i	j

Sol. The first node in preorder is a and hence a is the root node. unique. Binary tree is shown in fig 17

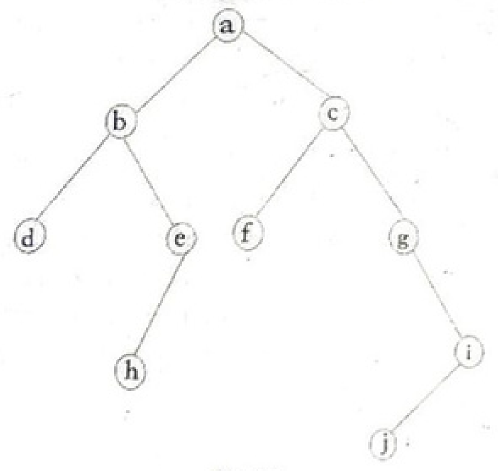


Fig.17

(a) Algorithm to Draw a Unique Binary Tree When Inorder and Postorder Traversal of the Tree is Given

1. We know that the root of the binary tree is the last node in its postorder. Draw the root of the tree.
2. To find the right child of the root node, first use the inorder traversal to find the nodes in the right subtree of the binary tree. (All the nodes that the rights to the root node in the inorder traversal are the nodes of the right subtree). After that the right child of the root is obtained by selecting the last node in the postorder traversal of the right subtree. Drwa the right child.
3. In the same way, use the inorder traversal to find the nodes in the left subtree of the binary tree. then the left child is obtained by selecting the last node in the postorder traversal of the left subtree.
4. Repeat the steps 2 and 3 with each new node until every node is not visited in postorder. After visiting the last node, we obtain the unique tree.

Example Draw the unique binary tree for the given inorder and postorder.

Inorder	:	4	6	10	12	8	2	1	5	7	11	13	9	3
Preorder	:	12	10	8	6	4	2	13	11	9	7	5	3	1

Sol. We know that the root node is the last node is postorder traversal. hence one is the root node. Now check the inorder traversal, we know that root is at the centre, hence all the nodes that are left to the root node in inorder traversal are the nodes of left subtree and all that are right to be root node are the nodes of the right subtree.

Now visit the next node from back in postorder traversal and check its position in inorder traversal, if it is on left of root then draw it as left child and if it is on right then draw it as right child.

Repeat the above process for each new node and we obtain the binary tree as shown fig .18

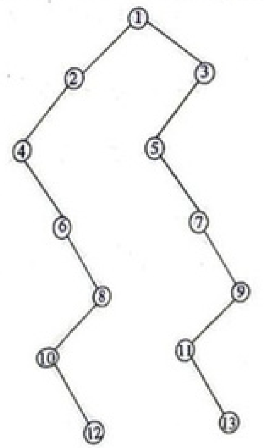


Fig.18

Example Draw the binary tree when inorder and postorder traversal is given:

Inorder : m k n j o l u s v q t p r

Preorder : m n k o u v s t q r p l j.

Sol. We know that the last node in postorder is the root node, hence j is the root. Now applying the algorithm as above, we obtain the tree shown in fig.19

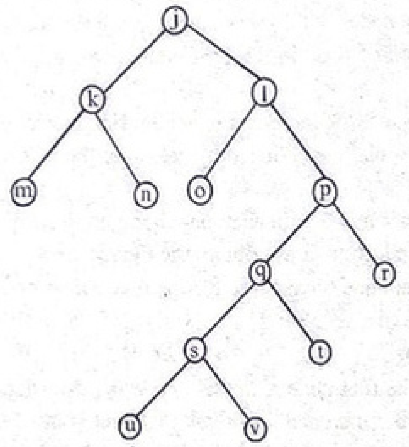


Fig19

(a) Algorithm to Convert General Tree into the Binary Tree

1. Starting from the root node, the root of the tree is also the root of the binary tree.
2. The first child C1 (from left) of the root node in the trees is left child Cl of the root node in binary tree and the sibling of the C1 is the right child of Cl and so on.
3. Repeat the step 2 for each new node.

Example 11. Convert the following tree as shown in fig.20 into binary tree.

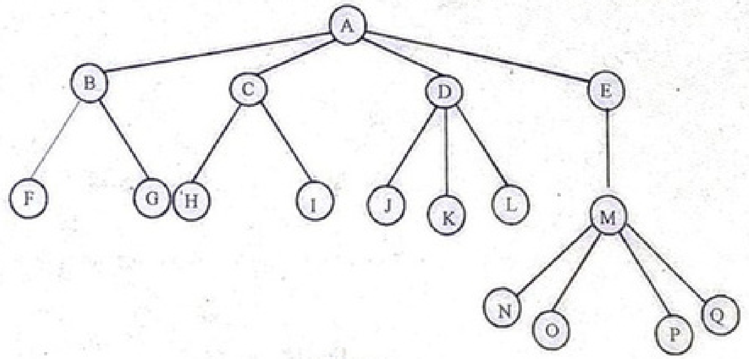


Fig.20

Sol. The root of the tree is the root of the binary tree. hence a is the root of the binary tree. now B becomes the left child A in binary tree, c becomes the right child of B, D becomes right child of C, and E becomes right child of D in the binary tree and similarly applying the algorithm we obtain the binary tree as shown in fig. 21

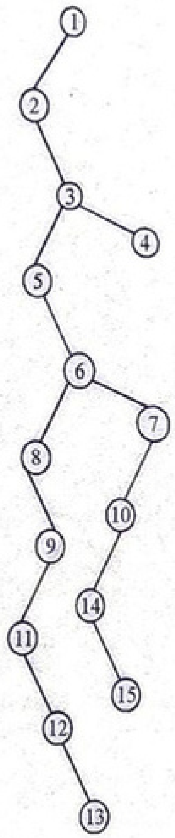


Fig 21

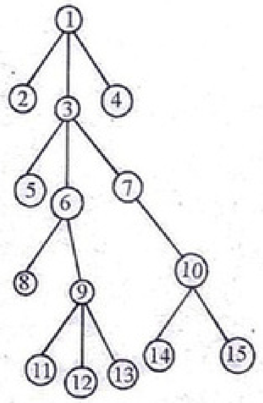


Fig. 22

Sol. The root node 1 in general tree is the root node of the binary tree. now applying the above we obtain the binary tree as shown in Fig.23

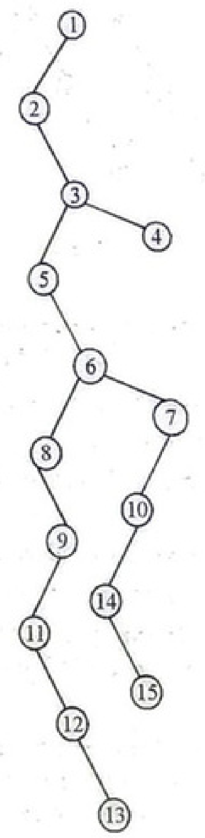


Fig 23

BINARY SEARCH TREES

Binary search trees has the property that the node to the left contains a smaller value than the node pointing to it and the node to the right contains a larger value than the node pointing to it.

It is not necessary that a node in a 'Binary Search Tree' point to the nodes whose values immedietly precede and follow it.

For example : the tree shown in fig. 24 is a binary search tree.

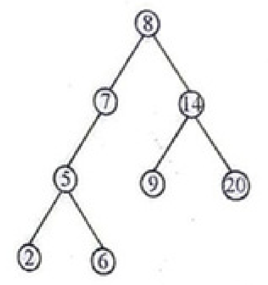


Fig.24

Inserting into a Binary Search Tree. Consider a binary search tree T. Suppose we have given an ITEM of information to inserted in T. the ITEM is inserted as a leaf in the tree.

The following steps explain a procedure to insert an ITEM in the binary search tree T.

1. Compare the ITEM with the roots node.
2. If ITEM>ROOT NODE, proceed to the right child and it becomes root node for the right subtree.
3. If ITEM<ROOT NODE, proceed to the left child.
4. Repeat the above steps until we meet a node which has no left and right subtree.
5. Now if the ITEM is greater than node, then the ITEM is inserted as the right child and if the ITEM is less than node, then the ITEM is inserted as the left child.

Deleting in a Binary Search Tree. Consider a binary search tree T. suppose we want to delete a given ITEM from binary search tree. to delete an ITEM from a binary search tree, we have three cases, depending upon the number of children of the deleted node.

1. **Deleted Node has no Children.** Deleting a node which has no children is very simple, as just replace the node with null.
2. **Deleted Node has only one child.** Replace the value of deleted node with the only child.
3. **Deleted Node has two children.** In this case, replace the deleted node with the node that is closest in the value to the deleted node. To find the closest value, we move once to the left and then to the right as far as possible. This node is called immediate predecessor. Now replace the value of deleted node with immediate predecessor and then delete the replaced node by using case 1 or 2.

Example Show the binary search tree after inserting 3, 1, 4, 6, 8, 2, 5, 7 into an initially empty binary search tree.

Sol. The insertion of the above nodes in the empty search tree is shown in fig.25

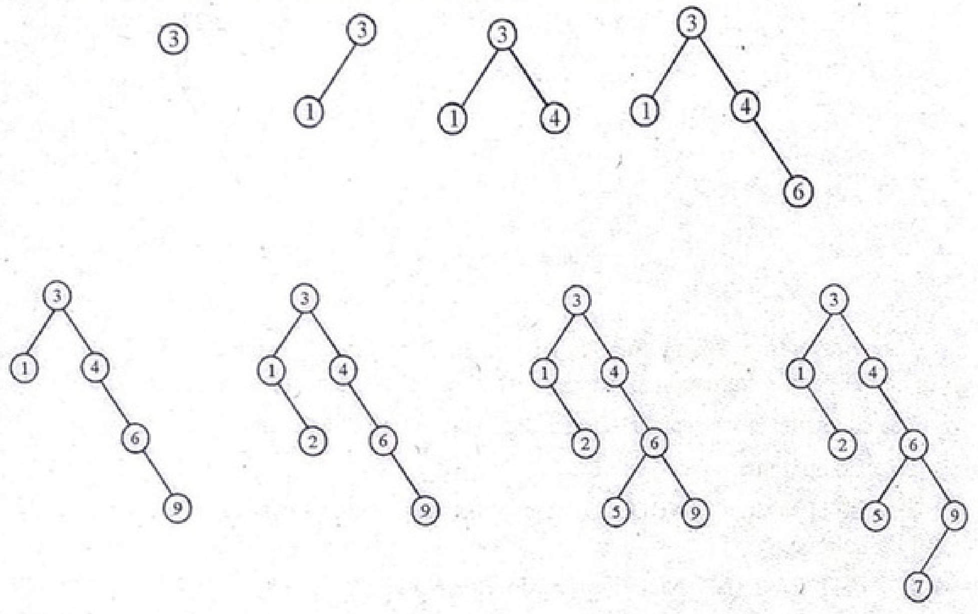


Fig.25

Example14. Show the binary tree shown in fig.25 (viii) after deleting the root node.

Sol. To delete the root node, first replace the root node with the closest element of the root. For this, first move one step left and then to the right as far as possible to the node. Then delete the replaced node. The tree after deletion is shown in fig 26.

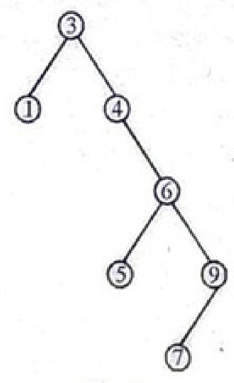


Fig.26

SPANNING TREE

Consider a connected graph g = (V, E). A spanning tree T is defined as a subgraph of G if T is a tree and T includes all the vertices of G.

Example. Draw all the spanning trees of the graph G shown in fig.27

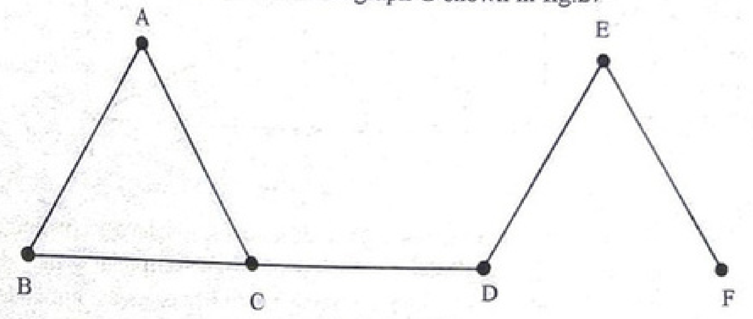


Fig 27 Graph G.

Sol. All the spanning trees of graph G is as shown in fig.28

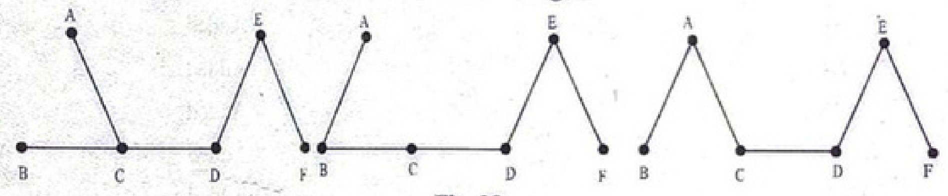


Fig 28

MINIMUM SPANNING TREE

Consider a connected weighted graph G = (V., E). A minimal spanning tree T of the graph G is a tree whose total weight is smallest among all the spanning trees of the graph G. the total weight of the spanning tree is the sum of the edges of the spanning trees.

The minimum weight of the spanning tree is unique but the spanning tree may not be unique because more than one spanning tree are possible when more than one edges exist having the same weight.

Kruskal's Algorithm to Find Minimum Spanning Tree. This algorithm finds the minimum spanning tree T of the given connected weighted graph G.

1. Input the given connected weighted graph G with n vertices whose minimum spanning tree T, we want to find.
2. Order all the edges of the graph G according to increasing weights.
3. Initialise T with all vertices but do not include any edge.
4. Add each of the graph G in T which does not form a cycle until n – 1 edges are added.

Example Determine the minimum spanning tree of the weighted graph shown in fig. 29

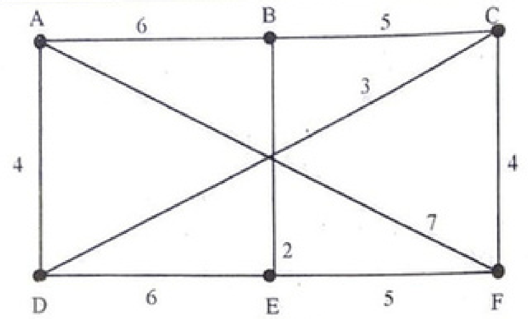


Fig. 29

Sol. Using kruskal's algorithm. Arrange all the edges of the weighted graph in increasing order and initialize spanning tree T with all the six vertices of G. now start adding the edges of G in T which do not from a cycle and having minimum weights until five edges are not added as there are six vertices.(fig.30)

Edges	**weights**	**Added or Not**
(B, E)	2	Added
(C, D)	2	Added
(A, D)	2	Added
(C, F)	2	Added
(B, C)	2	Added
(E, F)	2	Not Added
(A, B)	2	Not Added
(D, E)	2	Not Added
(A, F)	2	Not Added

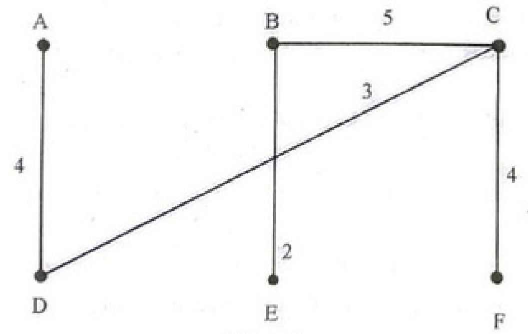


Fig. 30

Prim's Algorithm

In this algorithm we first choose a vertex v1 of the connected graph G and choose one of the edges having the smallest weight, which is not a loop and which is incident with v1, say e1 = (v1, v2). Now we choose an edge of smallest weight in G, which is incident with either v1 or v2, but at the end has neither of these two vertices. For example, we choose the edge e2 = (v1, v3), where i E (1,2), but v3=v1 or v2. We repeat the process of choosing the edges of smallest weight with the property that one of the end vertices of the edge is a vertex that has been previously chosen and the other end vertex appears for the first time, until all n vertices of G have been connected by edges, the resulting sub graph is a tree and so a spanning tree. the whole algorithm involves the following four steps:

Step1. Choose any vertex v1 of G.

Step2. Choose an edges e1 = (v1,v2) of G such that v2=v1 and e1 has the smallest

Step3. If the edges e1,e2......,ei have been chosen involving end vertices v1,v2....,vi+1, then chose an edge ei+1 =(vj,vk), with vj = (v1,v2....,vi+1), such that ei+1 has smallest weight among all the edges incident with the vertex in the vertex set (v1,v2....,vi+1)

Step4. If n-1 edges have been chosen, then stop. Otherwise go to step 3.

Example obtain a minimal spanning tree of the following weighted graph fig.31

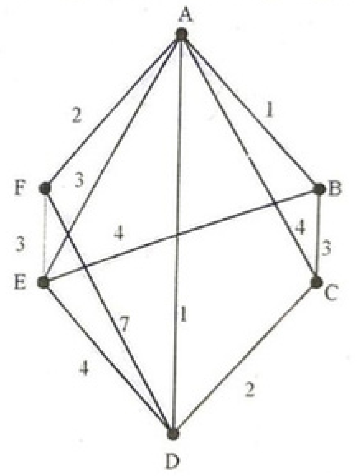


Fig 31

Sol. We perform the prim's algorithm, indicating at each stage, tge edges chosen, by heavy lines.

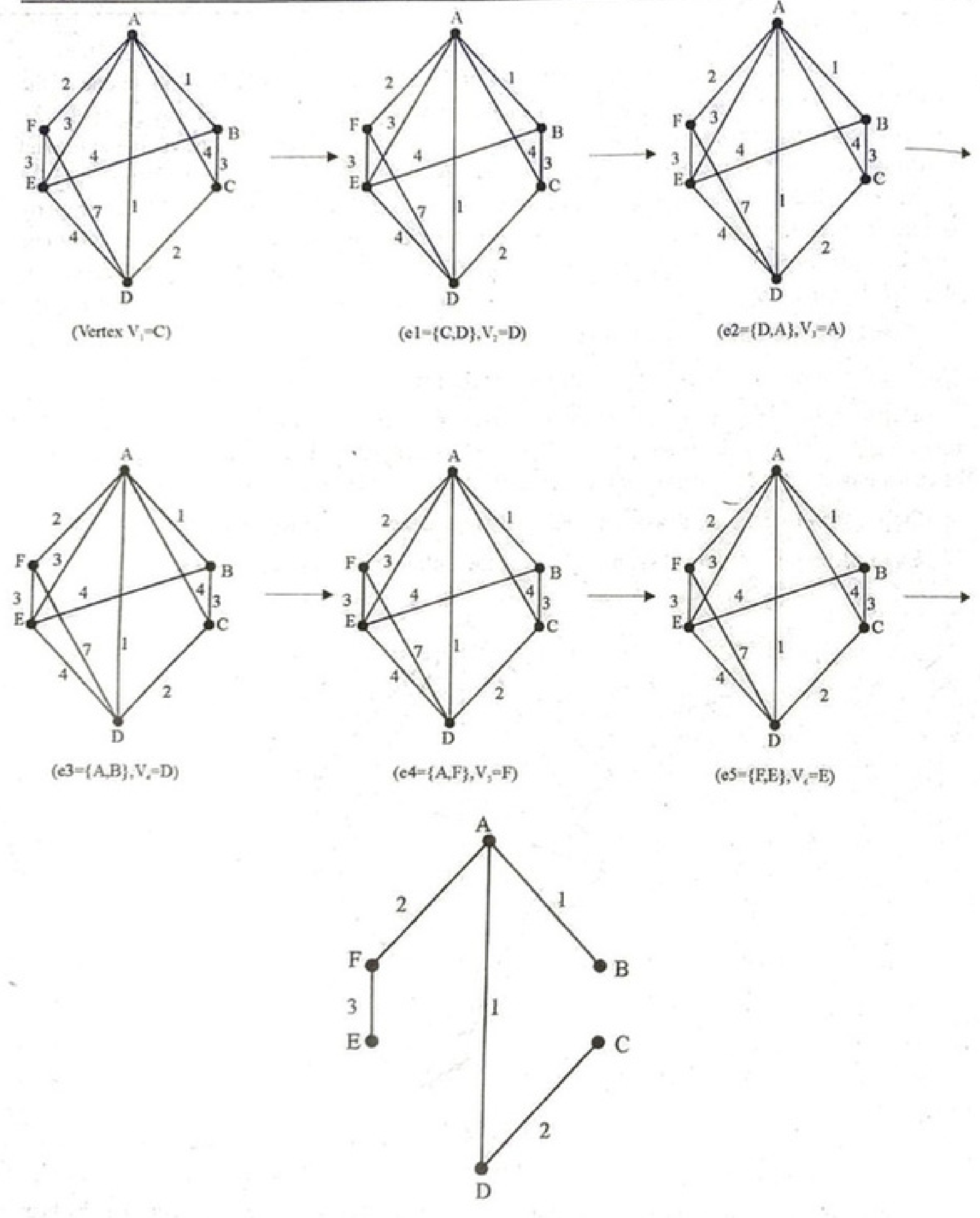


Fig 32 construction of minimal spanning tree using the prim's algorithm

GAME TREES

Trees are used in the analysis of two person games like chess, checkers and tic-tac-toe. In these games players alternate moves and each player tries to outsmart the opponent. The trees are used in the development of many computer programs that allows people to play against computers or even computers against computers. Here we discuss trees that are used to develop game playing strategies.

Example To understand the game trees in a better way. Let us consider a game tree as shown in fig. the values shown are obtained from an evaluation fuction.

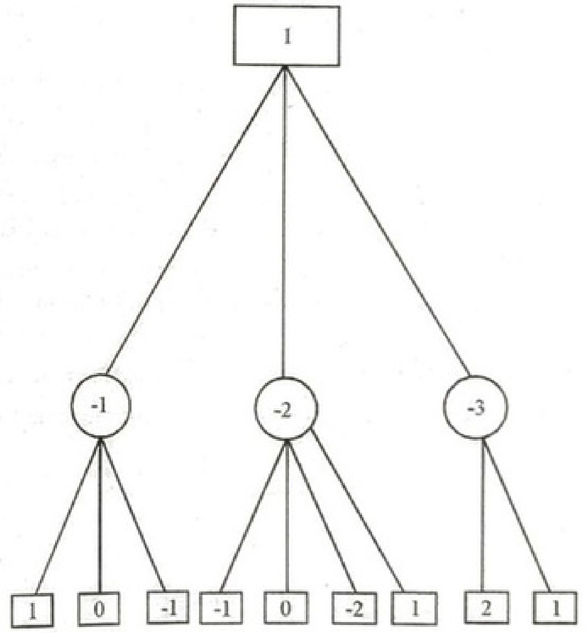


Fig 33.

The game tree is expended by two levels. The first player is represented by a box and the second player is represented by a circle. A path represents a sequence of moves. If a position is shown in a square, it is the first player's move. If a position is shown in a circle, it is the second player's move. A terminal vertex represents the end of the game. In our game tree, if the terminal vertex is a circle, the first player makes the last move and lost the game. If the terminal vertex is a box, the second player lost the game.

Minimax Strategy of Game Playing. It is a look ahead strategy. Here one player us called a maximize (who seeks maximum of its children) and the other is called a minimizer (who seeks minimum of its children).

Both the opponent, the maximize and minimizer fight it out to see that the opponent gets the minimum benefits while they get the maximum benefit.

A static evaluation function E is constructed that assigns each possible game position P the value E(P) of the position to the first player. After the vertices at the lowest level are assigned values by using the function E, the minimax strategy can be applied to generate the values of other vertices.

It is assumed that the static evaluation function returns a value from -10 to +10, where in a value of +10 indicates a win for the maximize and a value of -10 a win for the maximizer and a value of -10 a win for the minimizer. A value of 0 indicates a tie or draw. When using a game tree we should use a depth first search. If the game tree is so large that it is not feasible to reach a terminal vertex, the level to which depth-first-search is carried out can be limited. The search is said is said to be a K-level search if we limit the search to K-levels below the given vertex.

Example Apply the minimax to find the value of the root in the following game tree using a two level, depth-first minimax search. The values shown are obtained from an evaluation function.

Sol. Let us assume that the maximizer will have to play first followed by the minimizer. Before the maximizer. Moves to B, C and D, he will have to think which move would be highly beneficial to him.

i. If A moves to B, it is the minimizer who will have to play next. The minimizer always tries to give the minimum benefit to the other player and hence he will move to E. thus, the value – 4 us backed up at B.

ii. If A moves to C, then the minimizer will move to H and the value 0 is backed up at C.

iii. If A moves to D, then the minimizer will move to L and the value 1 is backed up at D.

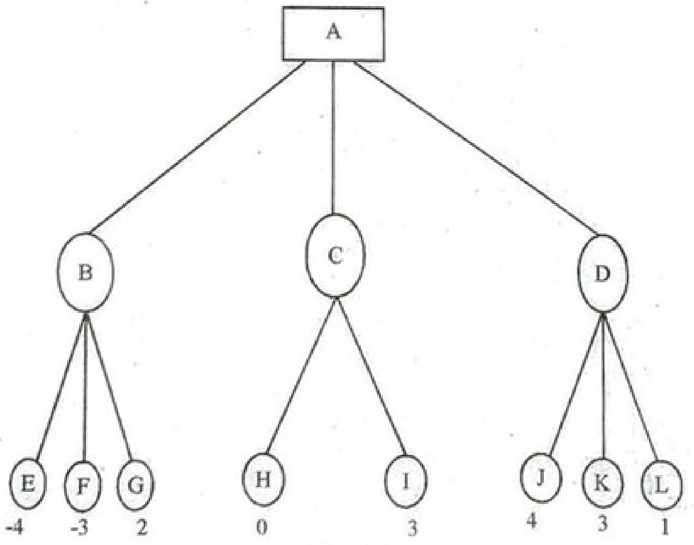


Fig. 34

The maximizer will now have to choose between B,C or D with the valuesw – 4, 0 and 1.The maximizer, will choose node D because it gives him a value of 1, that is better than – 4 and 0

The three with the backed up values is shown in fig. 35

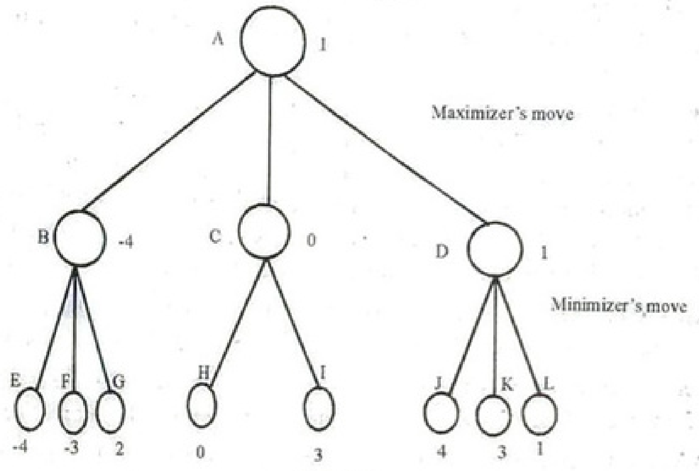


Fig 35

Now, assume that the minimizer will have to play first followed by the maximizer. In this case, the tree with the backed up values is shown in fig.36

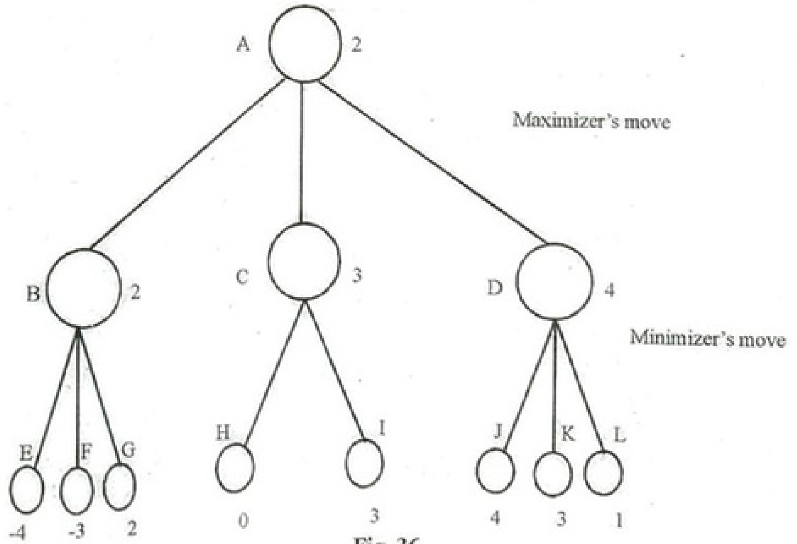


Fig 36

Example evaluates each vertex in the game tree shown in fig. the values of the terminal vertices are given below.

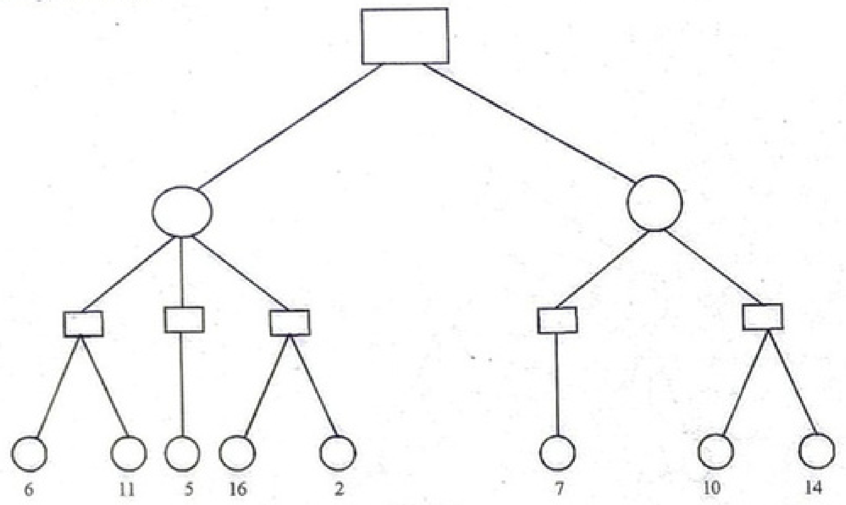


Fig 37

Sol. Let us assume that the maximizer will have to play first followed by the minimizer. The true with a threes level, depth-first minimax search is shown in fig.

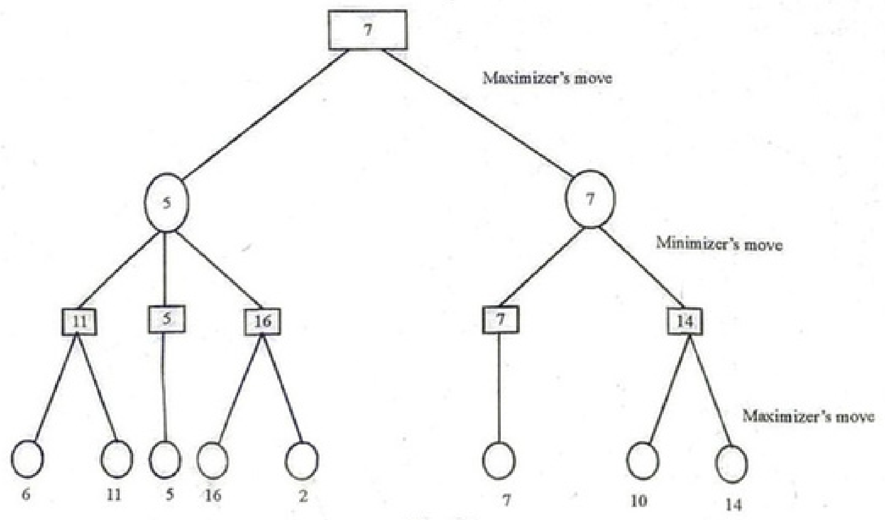


Fig 38

Minimax Strategy with Alpha-Beta Cutoffs. It is very time consuming to evaluate a game tree or a part of a game tree. the most common technique used to reduce the evaluation effort is called the alpha-beta pruning. This technique allows us to by pass many vertices in a game tree yet still find the value of a vertex. The value obtained is the same as if we had evaluated all the vertices.

Alpha cutoff. An alpha cut off occurs at a box vertex v when a grandchild w of v has a value less than or equal to the alpha value of v. the subtree whose root is the parent of w may be pruned. This pruning will not affect the value of v. The alpha value of a vertex is dependent on the current state of the search and changes as the search progresses. An alpha value for a vertex v is only a lower bound for the value of v.

Beta cutoff a beta cutoff occurs at a circle vertex v when a grandchild w of v has a value greater than or equal to the beta value of v. the subtree whose root is the parent of w may be pruned. This pruning will not affect the value of v. The beta value for a vertex v is only an upper bound for the value of v.

Example Consider the game tree shown in fig. Evaluate the value of vertex. A using a two level, depth first search with alpha cutoff. Assume that children are evaluated left to right.

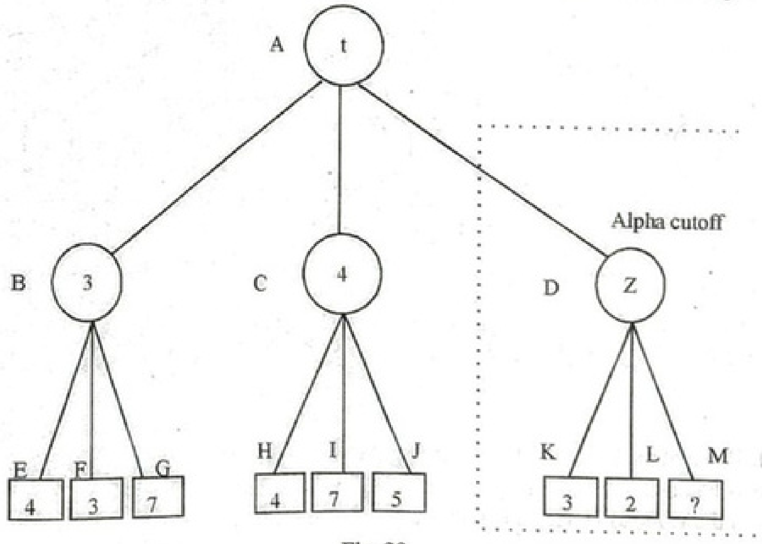


Fig 39

Sol Begin at the lower left by evaluating the vertices E, F and G. The value at vertex B is 3, the minimum of its children. Thus the value of t, at this point is at least 3, since the value at A is the maximum of its children. This is an alpha value of A.

The next vertices to be evaluated are H,I and J. The value at vertex C is 4, the minimum of its children.

The next vertices to be evaluated are K, L and M. when L evaluates to 2, we know that the value z of D cannot exceed 2, since the value of D is the minimum of its children.

From the above discussion, it follows that whatever the value of z is, it will not affect cutoff occurs. The value of T is 3 at A.

Example Evaluate the root of the game tree shown in fig., using a depth-first search with alpha beta pruning. Assume that children are evaluating left to right. for each vertex whose value is computed. Write the value in the vertex. Place a check by the roof of each subtree that pruned. The value of each terminal vertex is written under the vertex. Assume that the maximizer will have to play first.

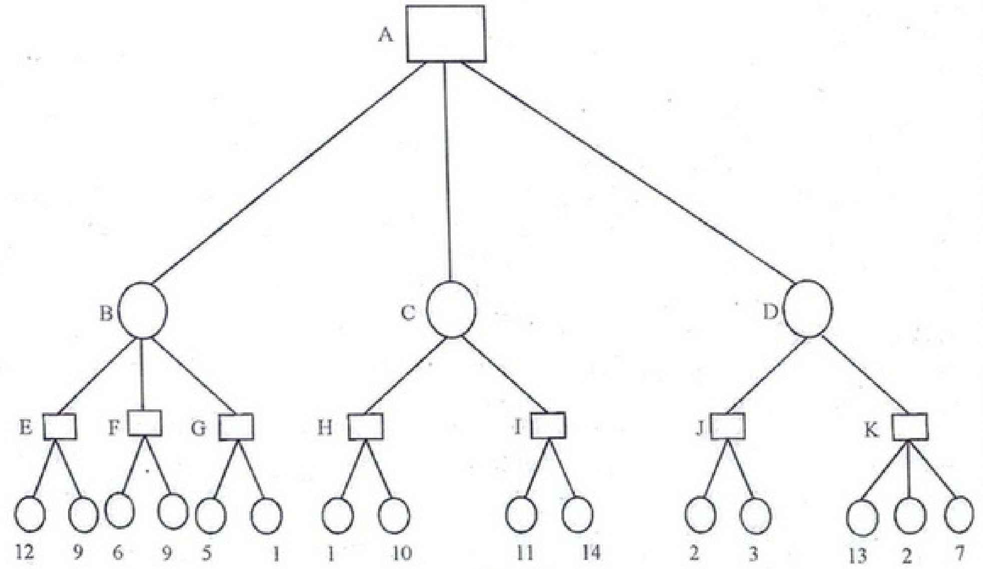


Fig 40

Sol. Begin at the lower left by evaluating the vertices L and M. The value at E is 12. Similarly the value at F and G is 9 and 5 respectively. The next we evaluate vertices R and S and the value of T is 11 and the value id U does not matter as next turn is of minimizer.

The next we evaluate V and W and the value of V is 3. The subtrees X, Y and Z are pruned after evaluating the value X, which is 13, as it is the turn of maximizer and the next turn, is of minimizer.

Now, we evaluate E, F and G. The value of B is 5. After evaluation of H and I. the values of C becomes 10 and the subtree I is pruned. Since it is the turn of minimizer. The subtree D is pruned since its value is 3 and the next turn is of maximizer.

Now, we valuate, B and C. the value of A is 10. The Complete tree with alpha-beta cutoffs is shown in fug.

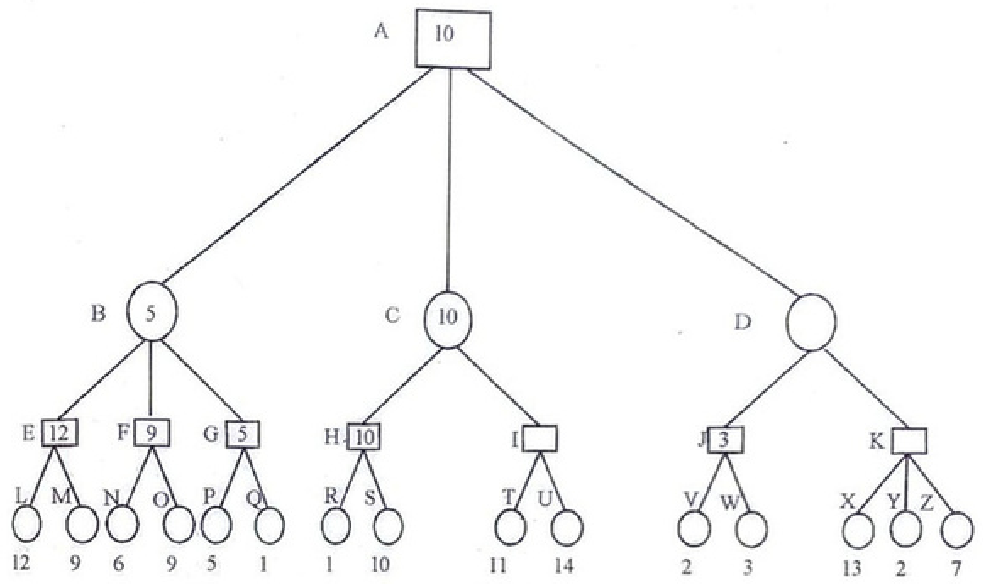


Fig 41

SOLVED PROBLEMS

Problem1. Show that if in a graph G there exists one and only one path between every pair of vertices, then G is a tree.

Sol. The graph G is connected since there is a path between every pair of vertices. A cycle in a graph exists if there is at least one pair of vertices (v1 ,v2) such that there exist two distinct paths from v1 to v2. But the graph G has one and only one path between every pair of vertices. Thus, G contains no cycle. Hence, G is a tree.

Problem2. Draw two different binary trees with five nodes having only one leaf.

Sol. The two trees out of many possible trees with five nodes having only one leaf is shown in fig.42

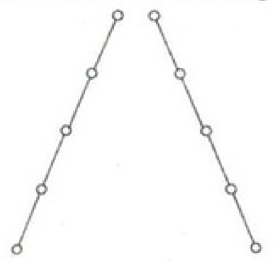
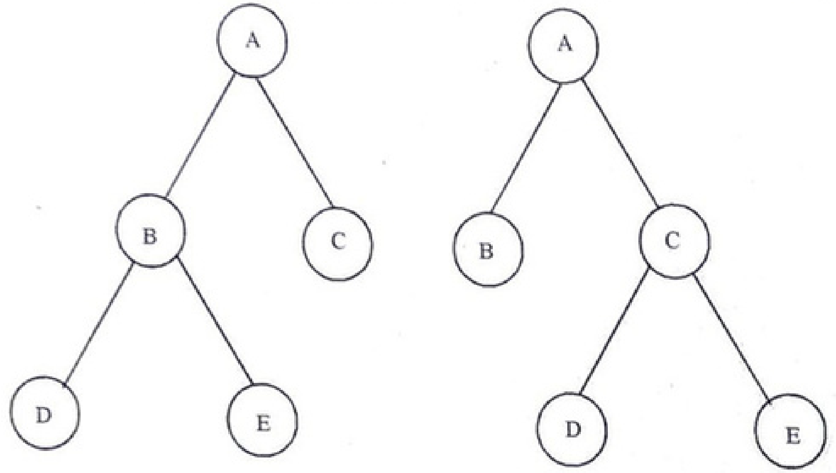
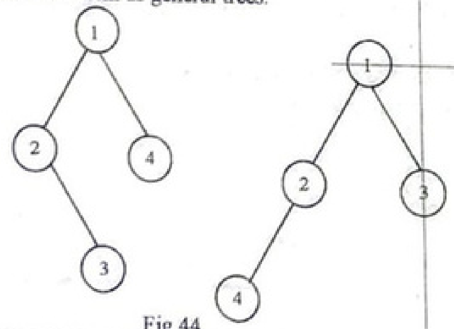


Fig.42

Problem3. Draw two different binary trees with five nodes having maximum number of leaves.

Sol. There are many possible trees, out of which two different binary trees are shown in fig 43.



General Tree	Binary Tree
1. There is no such tree having zero nodes or an empty general tree.	1. There may be an empty binary tree.
2. If some node has a child, then there is no such distinction.	2. If some nodes has a child, then it is distinguished as a left child or a right child.
3. The tree shown in fig.32 are same, when we consider them as general trees.  Fig 44	**3.** The trees shown in fig.32 are distinct, when we consider them as binary trees, because in (i), 4 us right child of 2 while in (ii), 4 is left child of 2.

Problem5. Determine the value of expression tree shown in fig45.

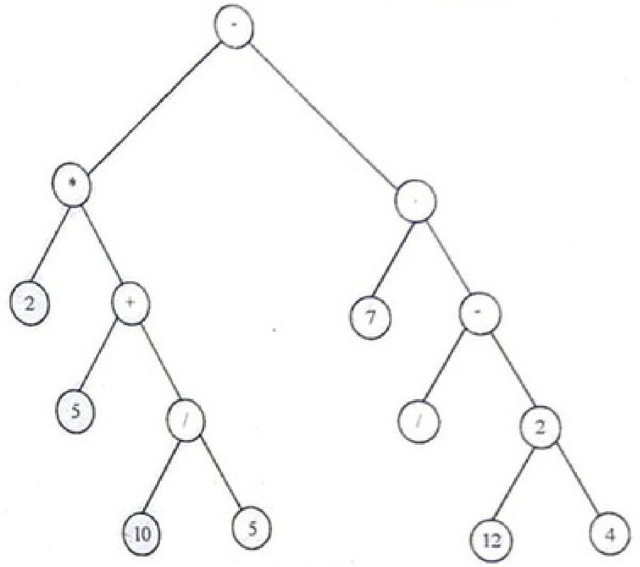


Fig 45

Sol. The value of expression tree is 2.

Problem6. Determine the value of expression tree shown in fig.46

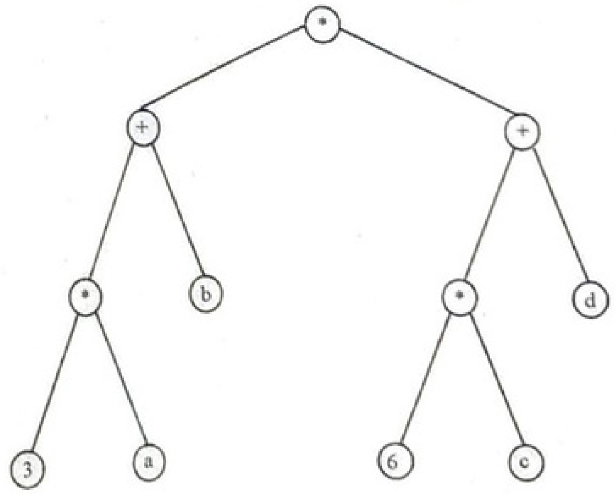


Fig 46

Sol. The value of expression tree is 7.

Problem7. Draw the unique binary tree when inorder and preorder traversal of tree is given as follows:

Preorder : 10 9 8 7 6 5 4 3 2 1

Inorder : 10 9 8 7 6 5 4 3 2 1

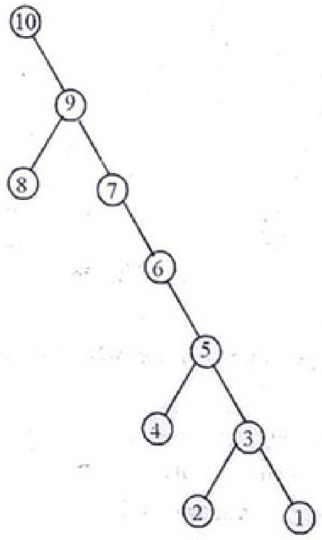


Fig 47

Sol. Unique binary tree is shown in fig 47

Problem8. Draw the unique binary tree when inorder and preorder traversal of tree is given as follows:

Inorder : 3 * a + * 6 * c + d

Preorder : * + * a b + * 6 c d

Sol. Unique binary tree shown in fig.48

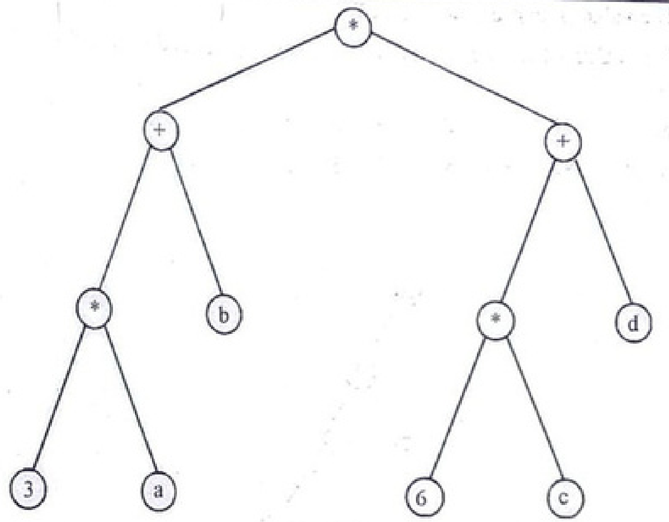


Fig. 48

Problem9. Draw the binary expression tree, when inorder and postorder traversal of the tree is given follows:

Inorder : k l -m + n p * q r t-/ + -*

Preorder : k - l+m * n - p + q /r - *

Sol. Binary expression tree is shown in fig.49

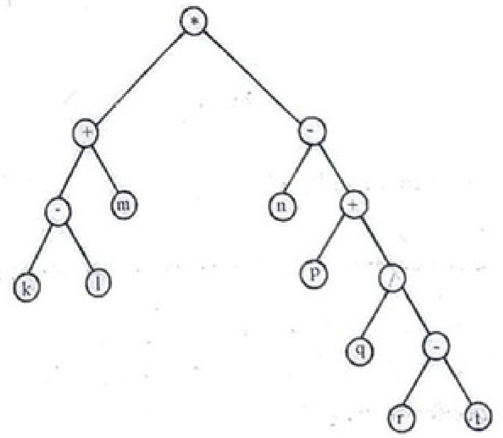


Fig .49

Problem10. Convert the forest shown in fig.50 in to binary tree.

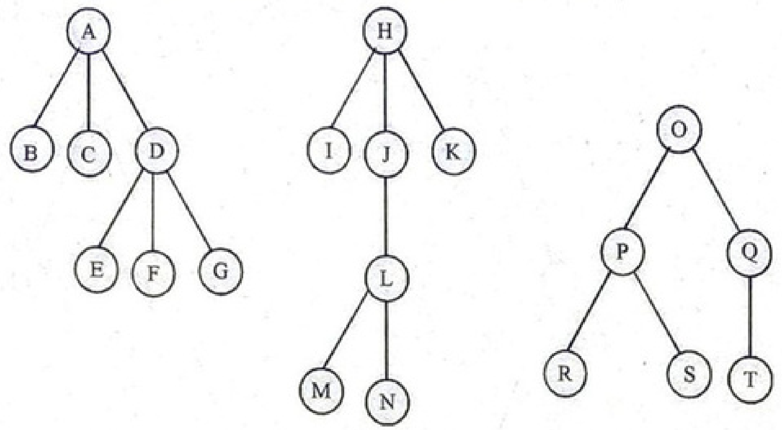


Fig 50

Sol. The root of the binary tree is the root node of the first tree (from left) and the root node of the second tree becomes the right son of the root node in binary tree and the root node.

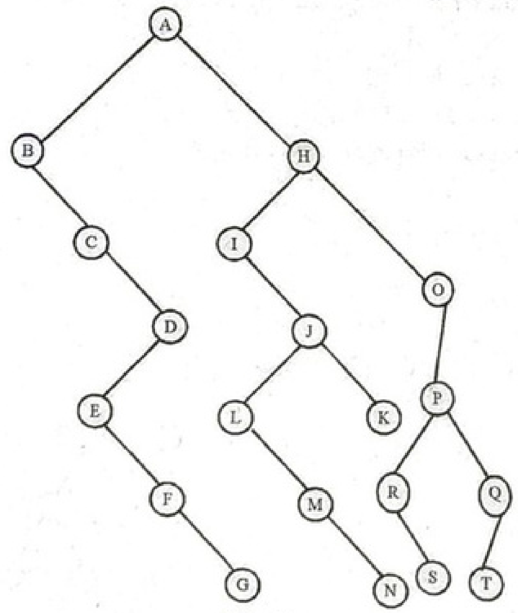


Fig 51

Of the third tree becomes the right son of the right son in binary tree. repeat this procedure for each level and we obtain the binary tree as shown in fig.51

Problem11. A binary search tree contains the value 1, 2, 3, 4, 5, 6, 7, 8, 9.The tree is traversed in pre order and the values are printed out. Determine the sequence of the print out values.

Sol. First of all draw the binary search tree as shown in fig.52

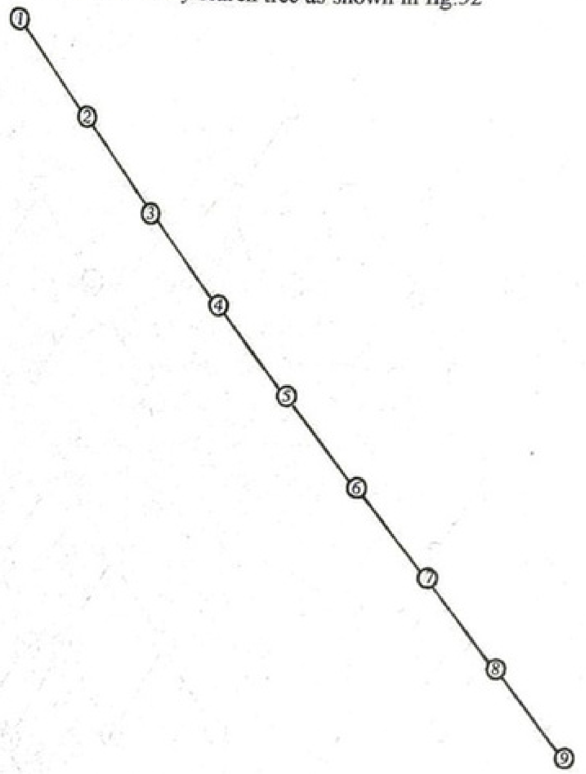


Fig 52

Problem12. A binary search tree is generated by interesting in order the following intevgers:

50, 15, 62, 5, 20, 58, 91, 3, 8, 37, 60, 24

Determine the number of nodes in the left subtree and right subtree of the root.

Sol. First of all draw the binary search tree as shown in fig.53

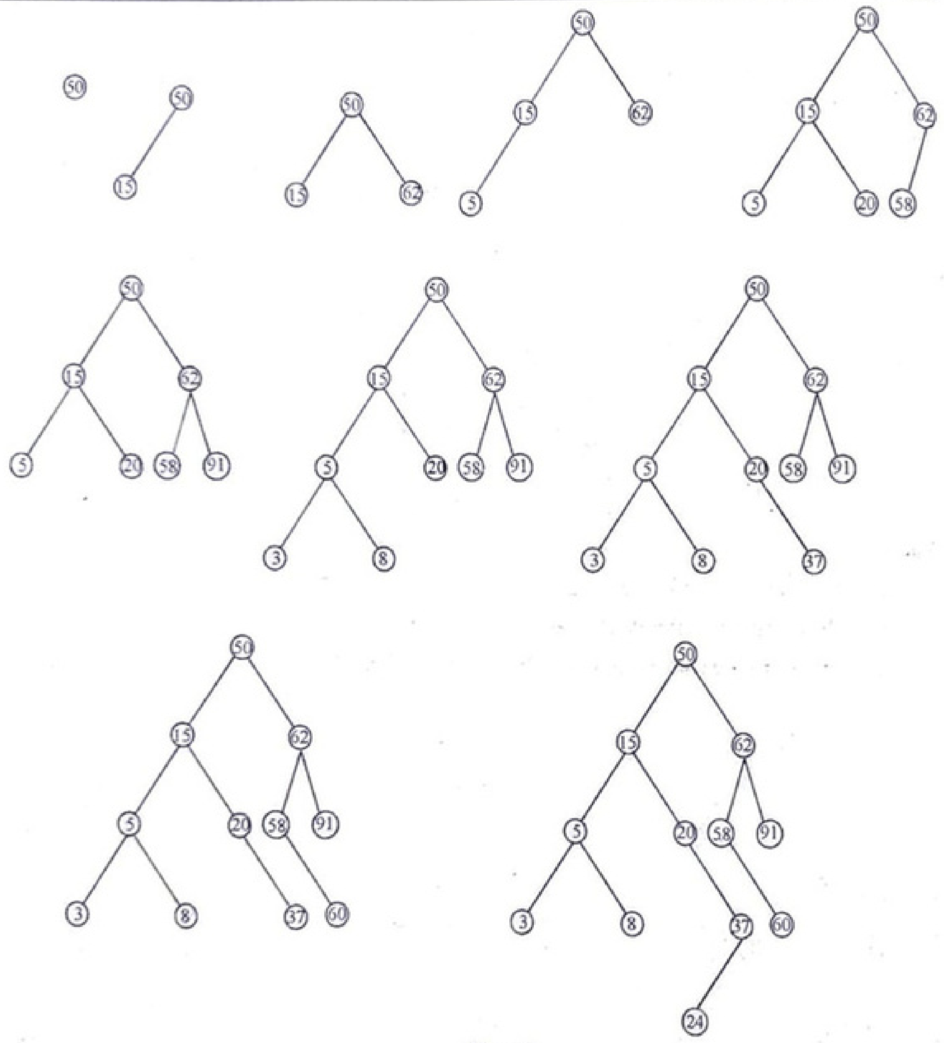


Fig. 53

Thus, the number of nodes in left subtree of the root is 7 and right subtree of the root is 4.

Problem13. Consider the binary tree as shown in fig.54. draw the binary tree for each of the following operations, if applied to the binary tree.

i. Delete the node V

ii. Delete the root node R.

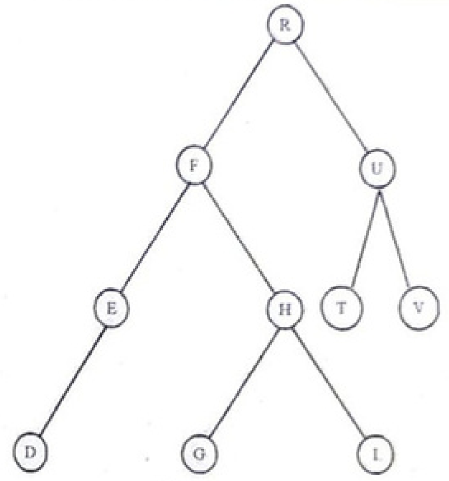


Fig 54

Sol.

i. The binary tree after deleting node V is shown in fig.55

ii. The binary tree after deleting node E is shown in fig.56

iii. The binary tree after deleting node R is shown in fig.57

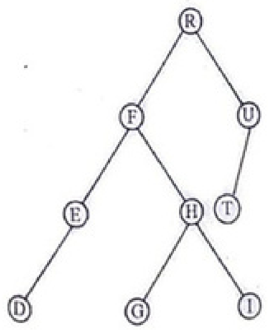


Fig. 55

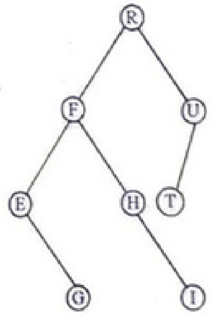


Fig. 56

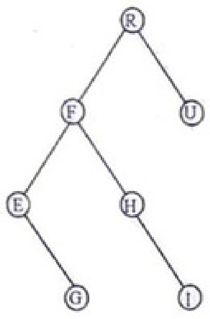


Fig. 57

Problem14. Find a minimum spanning tree of the labeled connected graph shown in fig.58

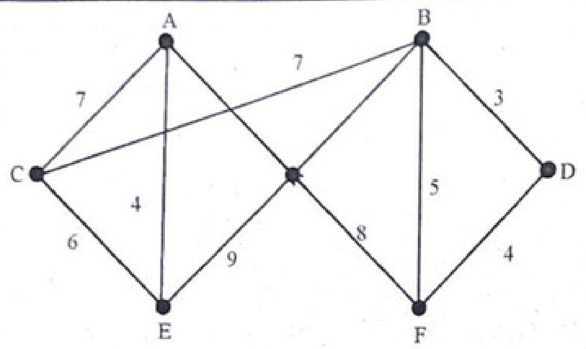


Fig 58

Sol. Using KRUSKAL'S ALGORITHM, arrange all the edges of the graph in increasing order and initialize spanning tree with all the vertices of G now, add the edges of G in T which do not form a cycle and have minimum weight until n-1 edges are not added, where n is the number of vertices. The spanning tree is shown in fig.59

Edges	**weights**	**Added or Not**	**Minimum Spanning Tree**
(B, D)	2	Added	
(A, E)	2	Added	
(D, F)	2	Added	
(B, F)	2	Not Added	
(C, E)	2	Added	
(A, C)	2	Not Added	
(B, C)	2	Added	
(A, F)	2	Not Added	
(E, B)	2	Not Added	

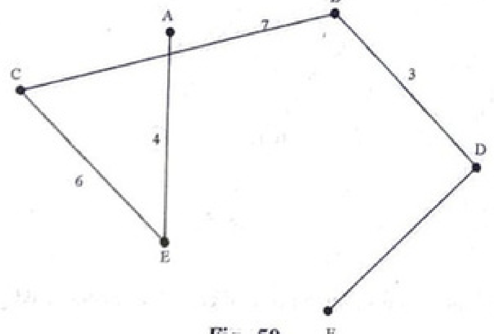


Fig. 59

The minimum weight of spanning tree is = 24.

Problem15. Find all the spanning trees of graph G and find which is the minimal spanning tree of G shown in fig.60

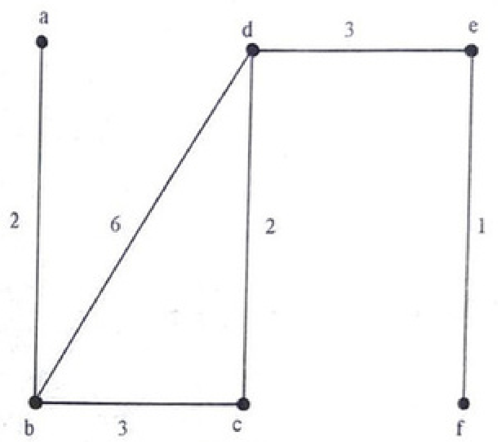


Fig 60

Sol. There are total three spanning trees of the graph G which are as shown in fig.61

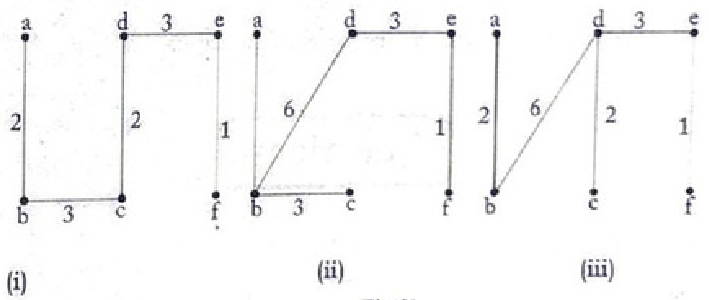


Fig 61

To find the minimal spanning tree, use the KRUSKAL'S ALGORITHM. The minimal spanning tree is shown in fig.62

Edges	**weights**	**Added or Not**	**Minimum Spanning Tree**
(E, F)	2	Added	
(A, B)	2	Added	
(C, D)	2	Added	

(B, C)	2	Added
(D, E)	2	Added
(B, D)	2	Not Added

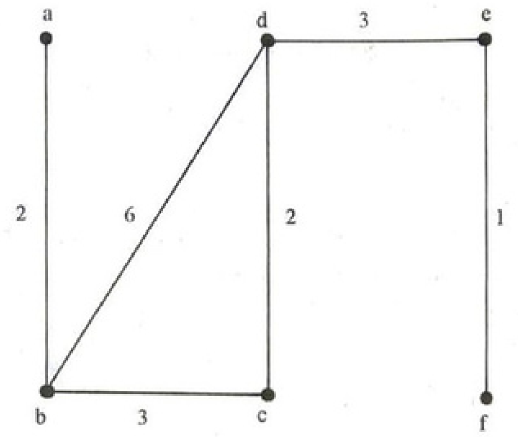


Fig. 62

The first one is the minimal spanning having the minimum weight = 11.

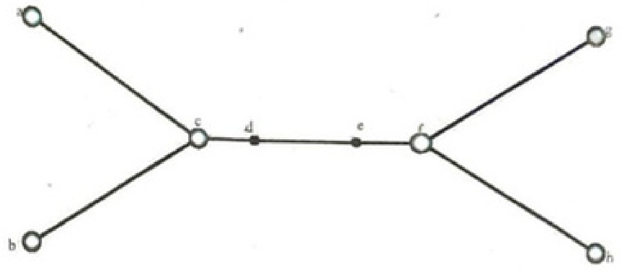
Questions

Very short:-

1. Define torest?
2. A tree with 5 vertexes has how many edges?
3. In a complete binary tree find number f maximum nodes at level 4?
4. How many centers a tree can have?
5. Draw BET for a + b.
6. What is MST?
7. Define siblings?

Short:-

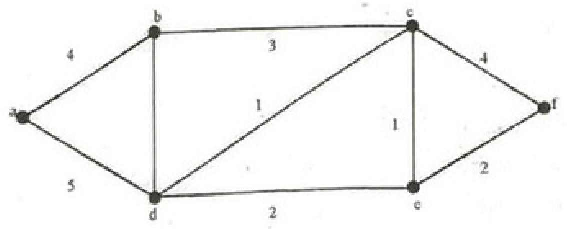
1. Find radius and center for following tree?



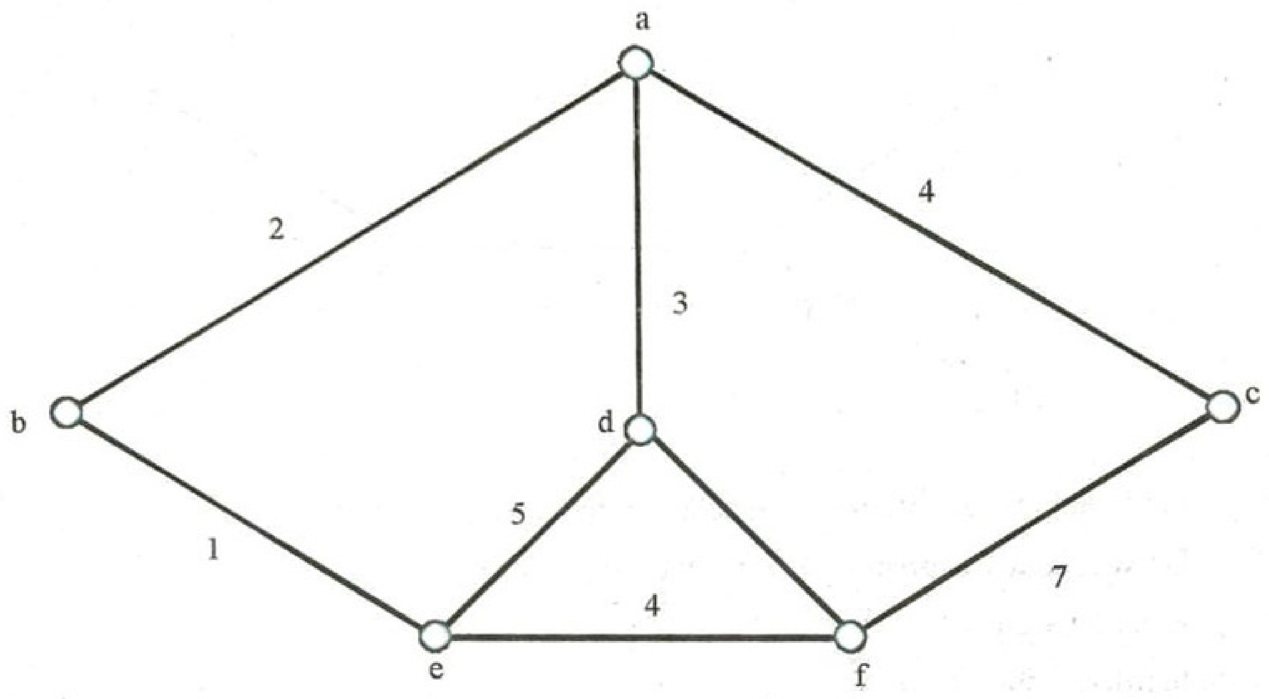
2. Find total number of maximum nodes up to level no 5 in a complete binary tree?
3. Draw a binary expression tree for the expression?
4. Preorder:- abdcet
   Inorder: - dbaect
   Find post order traversal?
5. inorder:- fcbeda
   Postorder: - fcedba
   Find pre order traversal?

Long Questions :-

1. Find MST for the following graphs by using kruskal's method?



2. Find MST for following graph by using prim's method?



3. Write a short note on game tree.
4. Prove that a tree with n vertex can have maximum 2 centers?
5. Explain tree traversal algorithms with example?

❑❑❑